ulm university universität
uulm

Ulm University | 89069 Ulm | Germany

**Fakultät für Ingenieurwissenschaften,
Informatik und Psychologie**

Institut für Künstliche Intelligenz
Institutsdirektorin: Prof. Dr. Susanne Biundo-Stephan

Pascal Tobias Bercher, geboren in Baden (Schweiz)

Kumulative Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

Hybrid Planning

— From Theory to Practice

2017

Amtierender Dekan:	Prof. Dr. Frank Kargl
Gutachter:	Prof. Dr. Susanne Biundo-Stephan
Gutachter:	Prof. Dr. Malte Helmert
Gutachter:	Prof. Dr. Jörg Hoffmann
Tag der Promotion:	4. Dezember 2017

Acknowledgement

I owe my largest gratitude to my supervisor Prof. Dr. Susanne Biundo who has given me the opportunity to work in a highly interdisciplinary research field that addresses up-to-date research questions that are of general relevance for our today's society. The research questions raised by this field are, on the one hand, motivated by problems that many people face during their daily life; on the other hand, answering these questions does not only focus on developing actual running systems, however. Instead, it also requires to address foundational theoretical research questions that are both interesting and important by themselves, and additionally form the basis for future foundational research in the respective field – *Artificial Intelligence (AI) planning.* I am thankful for any advice and support during the last years.

I want to thank Prof. Dr. Malte Helmert and Prof. Dr. Jörg Hoffmann for being my external expert witnesses of this thesis. I also thank Prof. Dr. Uwe Schöning, Prof. Dr. Birte Glimm, and Prof. Dr. Dr. Wolfgang Minker for agreeing being members of the board of examiners.

An innumerable number of quite controversial discussions with my former colleague Thomas Geier have influenced my work, which I appreciate a lot. Most notably, some of the most influential results of this thesis were done by us. I thank my former colleague Bastian Seegebarth for his contributions to the hybrid planning system Panda$_2$, which is used for the empirical analysis of many of the techniques that are part of this thesis. Relatedly, my colleague Gregor Behnke is the main developer of Panda$_3$, the successor of the previous system. Panda$_3$ is used as the basis for the most recent results. Countless discussions with him and my colleague Daniel Höller have further resulted in many publications showing foundational, but practically relevant, results in the field of AI planning – many publications are yet to be written by us and I am looking forward for further collaboration. I especially appreciate any feedback by Gregor and Daniel for this thesis. I also want to thank Dr. Ron Alford for a successful collaboration that resulted into many interesting publications, which are directly related to this thesis.

I also want to thank students that I was or am currently supervising. Eduard Bichel was the main developer of a graphical tool that serves as basis for displaying and investigating the plan generation process, which I used to debug Panda$_2$ and for presentational purposes. I thank Shawn Keen for assisting in the empirical evaluations of the data produced by Panda$_2$. Mario Schmautz continued his work and adapted it to be used by Panda$_3$. I am still thankful for his high discipline and reliability.

Lastly, I thank my family for supporting me over the years, most notably my mother Evelyn Preiser for any encouragement and backup I needed. I am also thankful for any emotional backup from my passed-away cat, Momo – to whom I dedicate my thesis.

Abstract

This work lays fundamental groundwork for the development of so-called *Companion Systems* – cognitive technical systems that are capable to reason about themselves, their users and environment, and to plan a course of action to achieve their users' goals. They are intelligent devices that assist their users in operating them: instead of the user having to learn how to operate the respective system, the system is intelligent and flexible enough to provide its functionality in a truly user-friendly way.

To fully meet a user's demands, *Companion Systems* rely on a multi-facet of capabilities that stem from different disciplines, such as Artificial Intelligence (AI) planning, knowledge representation and reasoning, dialog management, and user interaction management, to name just a few. This thesis focuses on the relevant aspects of AI planning technology that are of importance for such systems. AI planning is the central technology for many *Companion Systems* as it allows to compute a course of action that, if followed by its user, achieves his or her goals and therefore serves as a basis of providing advanced user assistance. This thesis is concerned with *hybrid planning* – a hierarchical planning formalism that is especially suited for the basis of providing assistance to human users. Based on this formalism we will investigate the full endeavor of developing *Companion Systems* – from theory to practice.

The thesis presents a novel formalization for hierarchical planning problems, which has become a standard in the field. We present a categorization of different problem classes into which hybrid planning as well as other well-known problem classes fall. This formalization allowed to prove a series of novel complexity results that are of interest both for theoretical and practical considerations. For many of the identified classes we introduce novel heuristics that are used to speed up the solution generation process. Some of them are the very first for the respective problem class, and some are the first admissible ones, thereby allowing to find optimal solutions – which is especially important when plans are generated for human users. We apply hybrid planning in a prototypical *Companion System*. It assists a user in the task of setting up a complex home entertainment system. Based on a declarative (planning) model of the available hardware and its functionality, the assistant computes a sequence of actions that the user simply needs to follow to complete the setup task. Several so-called *user-centered planning capabilities* are applied in this system, such as a technique for generating user-friendly linearizations of non-linear plans or the capability to answer questions about the necessity of actions – an essential property to ensure transparency of the system's behavior.

In conclusion: Most modern technical devices are still lacking true intelligence – since no research such as AI planning is sufficiently applied, so there is still huge potential in making such devices really smart by implementing them as cognitive systems that effectively assist their human users. Applying the research presented in this thesis is one step towards achieving this goal.

Table of Contents

To make clear which of the cited publications are *core contributions* of the thesis that are summarized in detail, we use just numbers to cite them; for *related work from the author*, the respective citations are preceded by an “A”; *related work from the literature* is preceded by an “L”.

Planning is the art and practice of thinking before acting.

Patrik Haslum [L50]

1 Introduction

Technical devices are becoming more and more complex. Usually, no single user is aware of the full spectrum of the functionality of a typical modern technical device. Even more importantly, many systems of our daily lifes do not focus on user-friendliness – at best, they have a well-structured menu or set of buttons, but the user still has to learn how to use them in order to achieve his or her goals. One of the few counter-examples are smartphones, for which assistants are available that allow to operate the phone or to execute certain web services like search engines via speech input. However, for most other every-day technical devices no scientific technologies, such as AI planning or dialog and user interaction management are being used to enhance the way in which they provide their functionality to the user. The underlying technology of most technical devices is still quite simple and does not at all exploit the full spectrum of today's scientific capabilities.

Companion Technology [3, L3, L9] aims at making technical devices really smart. Such systems, called *Companion Systems*, are equipped with a knowledge base storing relevant information about themselves, the application, the surrounding, and the user. Based on a multitude of capabilities stemming from different research disciplines, these systems are able to adapt to their current user and current situation and to assist him or her in a fully automated and user-friendly way. They do so by relying on domain-independent techniques, so that the resulting assistance functionality is not tailored to one specific application domain such as smartphones, but it can be deployed in many different contexts. Further details are provided in Sec. 4.1.

In this thesis we will address some of the most fundamental research questions related to pursuing the endeavor of creating *Companion Systems*. As Haslum stated, *"Planning is the art and practice of thinking before acting"* [L50]. Thus, planning is the pivotal means for goal-directed behavior of any kind of agent – be it human or artificial. Automated Planning, one of the main streams of AI, is concerned with finding a course of action that transforms a given situation into a desired one. It is easy to see that AI planning is a canonical way to enhance the functionality of technical devices, as they have a well-defined functionality: They are basically deterministic automata and can hence be modeled using *states* and *actions* – the most fundamental ingredients of planning. Partially ordered sets of actions form *partial, non-linear plans*. Those partial plans that are *executable* and achieve the problem's goals (i.e., that lead from an initial situation to one satisfying the goals) are called *plans* or *solutions*. These plans are used as the basis for *Companion Systems*: their actions can either control the behavior of the system itself or they are instructions that the user simply needs to follow.

Using AI planning as the heart of any *Companion System* requires to answer several research questions:

- Which planning paradigm is best suited as a basis for *Companion Systems*? What planning-related capabilities are important when interacting with a human user?
- What are the foundational theoretical properties of the chosen planning formalism? How can these properties be exploited in theory and practice?
- How can problems be solved quickly? Can we give optimality guarantees for the plans generated?
- Which practical issues do arise when constructing an actual *Companion System*?

We propose to use hybrid planning [4, 11] as the basis for providing user assistance (Sec. 4). Hybrid planning is a planning formalism in which standard *hierarchical task network (HTN)* planning [12, L56, L76] is fused with concepts from *partial order causal link (POCL)* planning [L57, L89].

Using this formalism provides several benefits when being applied in the presence of a human user. First, using a hierarchical formalism allows to model the application domain in a hierarchical manner. Presumably because hierarchical planning shows many similarities with the way in which humans solve their problems [L44, L51, L72], many real-world planning applications are modeled hierarchically [L10, L54, L56]. Using a hierarchical model further allows to use synergies between the planning model and a corresponding ontology thereby allowing, among others, to semi-automatically assist in the process of creating the model [A16]. Using a hybrid model in particular further facilitates the modeling process because hybrid models differ from standard HTN models in the way abstract tasks look like. In hybrid planning, these have preconditions and effects, which are used to further restrict the models in a way that coincides with the modeler's intent [4]. Second, hybrid planning allows the application of various user-centered planning capabilities. These are, among others, the generation of plan explanations [2, 6, 11, L28] as well as to repair plans that failed during execution [2, 6, 11, L41]. The former makes explicit use of the task hierarchy to generate justifications of any executed actions – using different levels of abstraction (Sec. 4.2).

First, in Sec. 2, the thesis lays the theoretical groundwork for our endeavor. For this, we give a classification of a wide spectrum of hierarchical problem classes, into which the hybrid planning formalism falls as well as several special cases that arise from posing additional restrictions. We present a novel formalization of these problem classes [12], which has already become an established standard, as it is regularly used and adopted both by us and other research groups working on hierarchical planning. We give a series of novel complexity results – both regarding the so-called *plan existence problem*

("How hard is it to decide whether there exists a solution?") [4, 12, A12, A13] and the so-called *plan verification problem* ("Given a plan and a planning problem, does the plan solve the problem?") [4]. Most notably, prior to this work, most theoretical results for hybrid planning were unknown and only a few were known for HTN planning. We transferred the latter as well as some novel results to hybrid planning [4]. The classification of these problems as well as their theoretical investigation lays the groundwork for a deeper understanding of these problem classes and their interconnections [A10, A20]. It is exploited in many different ways, e.g., for compilation techniques that transform these problems into each other [A7], for the construction of heuristics [1, 7, L18], or for addressing user requests to change plans according to their preferences [A8].

Given a model of the application domain and the problem that the user wants to solve, we need to investigate how such problems are solved practically. Especially when interacting with human users, it is of crucial importance that waiting times are reduced as much as possible. We are thus interested in how hybrid planning problems and their sub classes can be solved quickly. Answering this question is not only of practical relevance, but also theoretically interesting, because at the time being, there is only very limited research how to solve hierarchical problems automatically [1]. The thesis proposes an algorithm that is suited for generating solutions for humans quickly [7]. Its search procedure allows to easily incorporate the user into the plan generation process if he or she so desires, and – more importantly – it allows to generate optimal solutions if provided with admissible heuristics. For many of the investigated problem classes, we do provide heuristics [1, 7, 8, 9, 10], many of them are the first well-informed heuristics for the respective problem class [1, 7] or the first admissible ones [1, 8]. We analyze those heuristics both from a computational point of view and empirically. All matters concerning solving hybrid planning problems (and their sub classes) are investigated in Sec. 3.

So far, we have identified that we will use the hybrid planning formalism for providing advanced user assistance in the form of *Companion Systems*; we laid the theoretical groundwork by analyzing its computational complexity; we have specified an algorithm that is suited for solving these problems automatically; and we provided heuristics to speed up the solving process and that guarantee to find optimal solutions. We now show how these ingredients can be used and extended to be deployed in a prototypical *Companion System*, see Sec. 4. For this purpose, we first give a short introduction into the research vision and history of *Companion Technology* [3] and then explain the user-centered planning capabilities that were developed, improved, or deployed practically in the context of this thesis [2, 11]. We then introduce our assistant that helps a user to set up his or her home entertainment system [5, 6, A5, A22]. The system features components from several different research disciplines, including AI planning, knowledge representation and reasoning, and dialog and interaction management. Based on a declarative model of the available cables and devices, the system fully automatically generates a sequence of plug actions that its user simply needs to follow. At any time during execution, the user may ask questions about the purpose of the currently presented instruction. The system then generates – again, fully automatically – explanations, which do explain this purpose to the user. This explanation is presented in form of natural language (both written and spoken). At any time during execution, execution failures can be reported to the system, which will then derive a new plan that compensates the unexpected failures. We also present an empirical evaluation of the system with a special focus on the plan explanations.

The remainder of the thesis is structured as follows. The three lines of research summarized above – that is, *theoretical foundations*, *search and heuristics*, and the *practical application* of hybrid planning – are presented in Secs. 2 to 4. Each of these sections starts with a motivation for the main questions addressed in the respective section, followed by a brief summary of the main contributions including a list of summarized publications. We then explain these contributions in more detail and show how they fit into the field of research. Sec. 5 concludes the thesis and Sec. 6 lists the references. As mentioned before: To make clear which of the cited publications are *core contributions* of the thesis that are summarized in detail, we use just numbers to cite them; for *related work from the author*, the respective citations are preceded by an "A"; *related work from the literature* is preceded by an "L". In the beginning of Sec. 7, we give a table of contents for the core contributions of the thesis, which are appended in full length at the end of this document.

> *First, you have to understand the problem.*
> *After understanding, then make a plan.*
>
> George Pólya [L95]

2 Theoretical Foundations

Motivation. As Pólya stated: *"First, you have to understand the problem."* – i.e., the theoretical understanding of the given problem at hand should always be the very first step before attempting to solve it. After that very first step, we can choose an adequate formalism in which the given problem can be expressed. To pick the most-appropriate formalism in which the problem can be adequately represented, we also need a comprehensive understanding of the potential formalisms that may be chosen. We have already argued in the last section that we deem hierarchical planning especially suited for providing user assistance. Moreover, we choose *hybrid planning*, because it allows the deployment of several user-centered planning capabilities (see Sec. 4), and because it allows the automatic verification of the model during the stage of creating it (see Sec. 2.4).

Knowing about the theoretical foundations of a formalism is not only relevant for picking the most-suitable formalism, but also for a series of practical considerations. For one, the domain modeler is responsible for making sure that the *modeled problem* is actually representing (the practically relevant aspects of) the actual problem one wants to solve. If this was violated, the consequence can be that desired solutions cannot be generated or that solutions to the modeled problem are generated that do not correspond to solutions of the actual problem the model is supposed to represent (so-called "false witnesses"). Further, knowing about foundational properties of the respective formalism (and those related to it) often reveals manifold applications. For example, it often allows to give compilations that transform a problem expressed in one formalism into an equivalent problem expressed in another formalism. This is of practical relevance because it allows a rich language to be used by the domain modeler on the one hand, while the deployed techniques can focus on a lower-level description language on the other, so existent techniques do not need to be adopted after new language features are developed and used. It further allows to design specialized algorithms that exploit structural properties of the given problem (heuristics are prominent examples for this) or pruning techniques that speed-up the solution generation process, to name just a few.

Summary of Core Contributions. One of the most important contributions of this thesis is the development of a novel formalization of HTN planning problems [12]. As we will see later, there is a huge number of different hierarchical planning formalisms and only for one of them, a limited number of theoretical properties was known prior to this work. Since its publication in 2011, our proposed formalism has been the basis for a number of theoretical investigations, not only by our research group at Ulm University [4, A8, A10, A20, L2, L14], and in cooperation with other research groups [A7, A12, A13, A14], but also by others [L7, L8, L18, L26]. It was also influencing hierarchical planning formalizations that were used for purposes other than proving foundational properties [L11, L15, L16]. We hence consider it an established new standard for hierarchical planning. We will detail the strengths of this novel formalization in Sec. 2.3.

Another major contribution of the thesis is the investigation of the impact of *task insertion* on the computational complexity of the plan existence problem for hierarchical planning. In a nutshell, solving a hierarchical planning problem requires to refine an initial partial plan into an executable plan consisting only of primitive tasks. In typical hierarchical planning, this refinement process has to adhere strictly to the task hierarchy. That is, one is only allowed to apply so-called *(decomposition) methods* to modify a given partial plan – no tasks may be inserted into partial plans if they are not introduced as the consequence of applying a method. A method is simply a pair mapping an abstract task to pre-defined partial plan, which can be interpreted as an implementation (or "standard recipe") for that abstract task. Applying a method to its abstract task replaces that task by the method's partial plan and adds the constraints posed on the decomposed task. In our 2011 paper [12], we were the first to systematically investigate the theoretical impact of allowing tasks to be inserted into partial plans *arbitrarily* (i.e., independently of applying methods). We call the respective problem class *TIHTN planning* (*HTN planning with task insertion*). Theoretical properties of TIHTN planning were studied both by us [12, A10, A20], in collaboration with other research groups [A13, A14], but also independently of us [L7, L8, L18]. The general interest in this class is older than its first theoretical foundation [11, L61, L68, L71, L86], mostly due its practical advantages [2, L71]. One important reason for allowing task insertion is that it makes the modeling process easier. Due to task insertion, one can specify a model that is only *partially* hierarchical, where some actions that ensure executability can be inserted by the planner, so the modeler does not have to ensure this via exploiting the hierarchy alone. Our main result was that TIHTN problems are decidable – which is in contrast to HTN problems, which are only semi-decidable. We have shown that cyclic method applications are not required in TIHTN planning, which limits the influence of recursive method structures. This makes task insertion a relaxation for HTN problems that can be exploited to obtain (more) tractable heuristics [1, L18].

We extended our novel HTN planning formalism to obtain a new formalization for hybrid planning as well [4]. Hybrid planning extends HTN planning in two directions: First, abstract tasks also use preconditions and effects, which are used, among others, to restrict the set of decomposition methods to those that adhere specific criteria. Second, both the initial partial plan as well as all partial plans in the model (i.e., in the decomposition methods) may contain causal links – a concept known from partial order causal link

(POCL) planning [L57, L89]. We will give details in Sec. 2.2. Our novel formalization of hybrid planning allows to transfer many results from HTN planning to the hybrid setting. More specifically, we were able to transfer most complexity results for HTN planning concerning the so-called *plan existence problem* [12, A12, L76] as well as concerning the *plan verification problem* [L14] to hybrid planning [4]. In fact, prior to this work, *no* computational complexity results were known for hybrid planning. Now, (almost) all hardness results could be transferred (membership results remain future work).

Another contribution is the investigation of the impact of specifying preconditions and effects for abstract tasks [4]. Many researchers have used formalizations in which abstract tasks have preconditions and effects [11, L19, L20, L44, L48, L61, L62, L68, L79, L85, L86, L87, L90]. Some of them exploit these preconditions to pose additional constraints on the set of decomposition methods that may be specified for the respective abstract task. We call these constraints *legality criteria* (or, synonymously: *implementation criteria*). These preconditions and effects – in combination with the related legality criteria – can be exploited for modeling support, since a tool can make sure that decomposition methods are actual implementations of the given abstract task. This is especially important for the endeavor of applying planning technology in practice, as modeling the domain is always one of the very first steps. In our paper, we provided a survey on these criteria that were posed in the literature, provided another one, analyzed their impact on the computational complexity (as reported above), and discussed the practical benefits of these criteria for modeling hierarchical planning domains.

When investigating the model's structure and solution criteria of the problem classes discussed so far, we can obtain a clear classification of the different planning problems [A17]. The two essential dimensions are whether task insertion is allowed and whether the model may contain causal links (in the latter, abstract tasks also have preconditions and effects in combination with legality criteria). This results in four problem classes: HTN planning and hybrid planning, each with and without task insertion. If we further take into account whether the initial partial plan contains abstract tasks, we get two further problem classes: partial order planning (POP) problems and partial order causal link (POCL) planning problems (see Fig. 2.1 for all six classes and their relationship). This classification allows a systematic view on various planning formalisms and shows their interrelations. This, in turn, shows for which of these classes results are still missing. For the POP and POCL problem classes, for instance, only limited results were known prior to this thesis. We summarize our findings for these classes next.

Considering the POP and POCL planning problem classes allows to investigate the computational hardness of turning a search node consisting only of primitive tasks into a solution [A17]. This is of practical importance to come up with well-informed heuristics for search nodes in which there are no abstract tasks. This is both the case if the problem was specified in a non-hierarchical manner in the first place, or simply because all abstract tasks have already been decomposed to a primitive level, but no executable

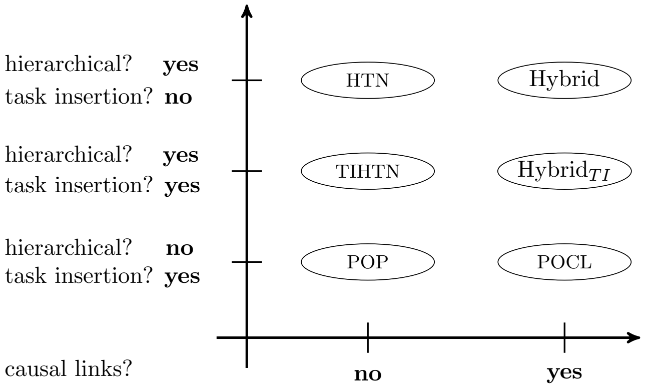


Figure 2.1: Overview of the investigated problem classes. The right side depicts the problem classes in which the initial partial plan or any partial plan of the model's methods may contain causal links. In the case of hybrid planning problems, abstract tasks also use preconditions and effects. The left side depicts the respective classes, where causal links are not part of the formalism. The bottom-most classes POP and POCL do allow task insertion. In the middle, task insertion is still allowed, but the initial partial plan may contain abstract tasks. On top, we see the hierarchical problems, where task insertion is prohibited.

plan was found yet. For such problems, we investigated the problem of partial delete relaxation, proving that the respective plan existence problem is $\mathbb{NP}$-complete. With partially delete-relaxed, we mean that all negative effects of the actions in the domain are ignored whereas the given (initial) partial plan remains unaltered [9]. Besides of providing more insights into the cause of computational hardness of such problems, that result directly laid to the development of a new heuristic that we investigate in Sec. 3.2.

The above-mentioned core contributions are published in the following publications:

[4] **P. Bercher**, D. Höller, G. Behnke, and S. Biundo. "More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks". In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. IOS Press, 2016, pp. 225–233. DOI: 10.3233/978-1-61499-672-9-225

[9] **P. Bercher**, T. Geier, F. Richter, and S. Biundo. "On Delete Relaxation in Partial-Order Causal-Link Planning". In: *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*. IEEE Computer Society, 2013, pp. 674–681. DOI: 10.1109/ICTAI.2013.105

[12] T. Geier and **P. Bercher**. "On the Decidability of HTN Planning with Task Insertion". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. AAAI Press, 2011, pp. 1955–1961

2.1 STRIPS Planning Problems

Before we start summarizing the thesis' contributions concerning the investigated problem classes, we first give some basic definitions to explain the most fundamental concepts of planning. In general, mathematical notation is made use of only occasionally to highlight some of the most important features. For a complete, coherent, formal description, please refer to the cited papers and articles.

In its most simple form, planning problems are given in a *non-hierarchical* way. The best-known (and least expressive) formalization is that of (propositional) STRIPS problems [L94]. All problem classes investigated later on can be regarded extensions of this formalism [A10, A20]. STRIPS problems are given in terms of a finite set of propositional *state variables* V, a finite set of *actions* A, an *initial state* $s_0 \in 2^V$, and a goal description $g = (g^+, g^-)$ with $g^+, g^- \subseteq V$. The state variables are used to model *states*, which are elements of 2^V – they describe how the world looks like in that respective state. The goal description g is a compact description of all states $s \supseteq g^+$ with $s \cap g^- = \emptyset$ that are considered *goal states* – they are the states one wants to end up in after executing a plan. *Actions* are used to transform states into each other. Actions are tuples $(pre^+, pre^-, \mathit{eff}^+, \mathit{eff}^-) \in (2^V)^4$ – they specify the positive and negative preconditions pre^+ and pre^- that need to hold (resp. are not allowed to hold) in a state in order for the action to be *executable* (or *applicable*) in that state as well as the positive and negative effects eff^+ and eff^- stating which state variables are added and removed from the current state. The objective is to find a sequence of actions that is executable in the initial state and ends up in a goal state. Often, one distinguishes between a so-called *domain model* $\mathcal{D}$, which describes just the underlying "physics" of the application scenario, and the actual *problem instance*[1] $\mathcal{P}$ one wants to solve (which references the domain). In STRIPS, we get $\mathcal{D} = \langle V, A \rangle$ and $\mathcal{P} = \langle \mathcal{D}, s_0, g \rangle$. Such STRIPS planning problems are ordinarily referred to as *classical planning problems*.

Due to the relative simplicity of non-hierarchical planning problems, there is a general consensus about how such problems syntactically look like and what semantics they underly. More specifically, such problems are often specified either in variants of the STRIPS formalism as presented above, or in *SAS*$^+$ [L78], where state variables are not propositional, but multi-valued (i.e., each variable has a finite set of possible values). The success of the STRIPS formalism has further been fostered by the planning domain description language *PDDL* [L66], which has been developed for the first International Planning Competition in 1998 and that was continuously extended to include additional language features. Such features are outside the scope of this thesis, however. That is, we always assume full observability and deterministic effects – similar to STRIPS. We further assume a ground (i.e., propositional, non-lifted) representation. We want to note

[1] The problem instance is also often referred to as *planning task*. We are not using that terminology here to avoid confusion, since in hierarchical planning the term *task* refers to actions.

that we only do so for the sake of readability. In most of the presented publications, we have investigated a lifted problem class.

We will start with the non-hierarchical problem classes POP and POCL, then investigate HTN and TIHTN problems, and finally consider the fusion thereof, hybrid planning.

2.2 POP and POCL Planning Problems

One of the reasons to investigate the computational complexity of a specific problem class is to come up with (tractable) problem relaxations that can be used as a basis for heuristics during search. The implicit assumption behind this motivation is that the search nodes of the deployed algorithm are of the same structure than the problem class itself.

For instance, all HTN planning algorithms that solve the respective problem via search perform this search in the space of partial plans – and the problem description is also given in terms of an initial partial plan (as we will see in Sec. 2.3), so investigating the computational complexity of HTN planning problems directly shows us the hardness of turning a *search node* in HTN planning algorithms into a solution. The similar situation is given for solving classical planning problems: the de-facto standard algorithm for solving such problems performs progression search in the space of states[2] – and the problem description is also given in terms of a state, so again complexity results for classical planning problems can be exploited for the search nodes of these procedures.

For POP or POCL planning procedures, the situation is different. To the best of our knowledge, the computational complexity of POP and POCL *problems* was not investigated in detail prior to this work. The reason for this lies in the fact that the literature only considered POP and POCL *algorithms* for solving classical planning problems, but the plan existence problem for a given partial plan was not investigated. So, one of the thesis' contributions is to investigate these problem classes as a basis for a deeper understanding required to come up with heuristics for the respective problem class/search node. As a consequence from these investigations, we have developed a novel POCL heuristic as well as a technique that allows to use heuristics known from state-based forward search in POCL planning. We will present them in Sec. 3.2.

Investigating these classes is not only important from a purely scientific point of view, or for the development of heuristics for POP and POCL search procedures, but also for solving hierarchical problems: When solving TIHTN problems or hybrid problems with task insertion, partial POP and POCL plans are produced after every abstract task has

[2] We will use the terms *forward search* and *progression search* synonymously.

been decomposed. So, applying standard POP and POCL heuristics to these search nodes is (likely to be) more informed than using heuristics that are mainly focusing on the abstract tasks rather than on the task insertion aspect.

POP and POCL Planning Systems (Related Work). There are various algorithms to solve classical planning problems. The best-known is forward search using primitive grounded actions, which is therefore also referred to as *classical planning*. In this approach, search is done in the space of states that are reachable from the initial state by applying executable actions. The search space is a tree (or a graph, in case duplicates are eliminated), its nodes are states, and its arcs are actions. The search ends successful as soon as a goal state is found. Then, a solution can be extracted from a path leading from the tree's root node to a goal leaf node.

Before several well-informed heuristics made this the superior planning approach, a very popular approach was planning in the space of partial plans [L91]. Here, each search node corresponds to a partial plan, i.e., to a partially ordered set of uniquely labeled actions – so-called plan steps (unique labeling is required to differentiate different occurrences of the same action). The search space is again a tree (graphs are ordinarily not maintained [L26], probably because it is computationally too hard to check whether two partial plans are isomorphic [L14]), its nodes are partial plans, and its arcs are modifications to these partial plans. Modifications to partial plans are, e.g., inserted actions or inserted orderings. The search ends successful as soon as a solution plan is found. In this search approach, each partial plan is partially ordered and regarded a solution if *every* linearization of its actions that is compatible with the ordering constraints forms a solution.

For this, some early partial order planners like TWEAK [L91] and UA [L82] insert constraints into a plan until every action precondition is necessarily true. To make the commitment, which precondition is necessarily true, explicit, the concept of so-called *causal links* was developed [L88, L89]. A causal link is a 3-tuple (ps, φ, ps') consisting of the producer plan step ps, the consumer plan step ps', and the protected condition φ, which is a precondition of ps'. Plan steps are uniquely labeled actions (again, the unique label is necessary to differentiate multiple occurrences of the same action from each other) and φ is a literal, i.e., a state variable $v \in V$ that is either positive in case it is a positive precondition of ps' and a positive effect of ps, or it is negative in case it is a negative precondition of ps' and a negative effect of ps. Action preconditions that are not protected by a causal link raise so-called *open precondition flaws*, and POCL planners insert causal links and required actions until no such flaws exist. Such planners further raise a so-called *causal threat flaw* when ever the partial ordering allows a plan step ps'' to be ordered between two plan steps ps and ps' sharing a causal link (ps, φ, ps') in case ps'' has an effect $\neg\psi$, such that ψ and φ can be unified. This causal threat represents the violation of the commitment that the protected condition φ is guaranteed to hold after the execution of ps until ps' was executed. It can be resolved by adding ordering constraints (ordering ps'' either before ps or after ps') or by adding a variable constraint that prevents that ψ and φ can be unified [L88, L89]. Partial plans that do not raise any flaw are considered solutions. It is easy to see that any linearization of a solution's plan steps corresponds to an executable action sequence in the standard STRIPS sense. The before-mentioned planners TWEAK and UA do not (yet) rely on causal links, however. So, planners of this kind are also referred to as non-causal link planners [L80]. Analogously, planners that do rely on causal links, such as SNLP [L89], UCPOP [L88], RePOP [L64], VHPOP [L59], or CPT [L52] are referred to as partial order causal link (POCL) planners [L80].

Complexity Results. We now interpret a given POP or POCL search node as a problem of the respective class. That is, instead of having just an initial state and a goal description, we have given an arbitrary (but finitely large) initial partial plan. As in STRIPS, the planning domain is given by $\mathcal{D} = \langle V, A \rangle$. In contrast to STRIPS, where the problem is given by a tuple $\mathcal{P} = \langle \mathcal{D}, s_0, g \rangle$, we now have $\mathcal{P} = \langle \mathcal{D}, P_I \rangle$, where P_I is an initial partial plan (that contains two special actions encoding the initial state s_0 and the goal description g). We call the respective problem class POP in case P_I does not contain causal links and POCL in case it does. It is easy to see that these classes subsume STRIPS problems as before, since we can simply define an initial partial plan that is empty except for the two artificial actions that encode the initial state and the goal description. A subtle difference between an actual STRIPS problem and a STRIPS problem encoded as POP/POCL problem is the solution criterion. In STRIPS, solutions are totally ordered action sequences, whereas solutions to POP and POCL planning problems may also be partially ordered (as long as every linearization is a solution). This does, however, not have any impact on the computational complexity (also, any totally ordered solution can be turned into a partially ordered one [L30]).

It is easy to see that POP and POCL problems have the same computational complexity as classical problems, i.e, they are $\mathbb{PSPACE}$-complete. Hardness is trivial, since the two classes extend classical problems. Membership can be shown in different ways, e.g., via translating a given partial plan into a classical problem [8][3]. A more direct proof consists of the following steps: First, guess a linearization of the partial plan's actions as well as complete states right before these actions (that are compatible with their preconditions). Then, apply the actions to these states, which gives us new (initial) states. This results in $n + 1$ classical problems for n actions in the partial plan, the solutions of which can be concatenated to a solution of the entire problem. Then, in each sub problem, reduce the action set according to the causal links. Solving the resulting problems each in $\mathbb{PSPACE}$ gives us an $\mathbb{NPSPACE}$=$\mathbb{PSPACE}$ decision algorithm. Thus, both POP

[3] In our paper, we only gave a polynomial translation for partial plans *without* causal links. Our translation for causal links is fixed-parameter tractable and can thus not serve for the reduction [8]. Meanwhile, we have also found a polynomial time compilation, but did not yet publish it.

and POCL problems have the same computational complexity as classical planning – which is non-surprising, since they stem from algorithms used to solve classical planning problems. While this is the case, it is still not clear whether problem relaxations show a similar behavior. Consequently, we investigated the plan existence problem for one of the most successful problem relaxations known from classical planning: delete relaxation.

When no action in the model uses negative preconditions (this can easily be achieved by a compilation technique [L73]), then ignoring the negative effects of the available actions is a problem relaxation that makes the plan existence problem for STRIPS problems decidable in $\mathbb{P}$ [L81]. This observation was exploited by the successful and therefore well-known (progression-based) FF planning system and its delete relaxation-based FF heuristic [L63]. The similar problem relaxation was never studied in POP or POCL planning. On the one hand, there are two well-known heuristics for POCL planning that *do* rely on delete relaxation (and can also be computed in $\mathbb{P}$), but on the other hand they also relax the delete effects of the given partial plan and they are also ignoring its causal links to a large extent [L59, L64] – and it was not clear whether these additional relaxations are actually necessary to obtain a tractable problem. There is a subtle but important difference about what exactly is relaxed in classical planning and by these two heuristics: in state-based forward search, the entire search progress is encoded in the current state. Thus, delete-relaxing the available actions has no impact on anything that was already achieved, i.e., the current state is not affected. In POP and POCL planning, it is the current partial plan that encodes the search progress, so relaxing parts of it means to also ignore information that was already obtained during search. We were thus investigating whether just ignoring the delete-effects of the available actions in the domain while not relaxing the current partial plan results in a tractable problem class as well. Interestingly, this is not the case: While this problem is in $\mathbb{P}$ for classical problems, it turned out to be $\mathbb{NP}$-complete for POCL planning [9, Thm. 1] (technically, we only showed this result for POCL planning problems, but it can easily be transferred to POP problems, because the proof does not inherently rely on causal links). To show the result, we adopted a proof by Nebel and Bäckström [L84, Thm. 15]) that shows $\mathbb{NP}$-hardness of deciding whether a partially ordered set of actions does possess an executable linearization of its tasks (a similar proof was given independently by Erol et al. [L76, Thm. 8]). All these proofs reduce from the $\mathbb{NP}$-complete SAT problem [L92, Thm. 13.1]. We can thus observe that the source of hardness stems from the problem of finding such an executable action sequence – and this problem does, as shown by our result, not become easier when we are allowed to insert delete-relaxed actions.

From this result we can conclude that it is actually required to perform delete relaxation for the given initial partial plan in order to come up with a tractable problem. On the other hand, as argued above, that partial plan encodes the progress of the search, so it might also be beneficial not to ignore this information and to cope with the $\mathbb{NP}$-completeness. In Sec. 3.2, we will see some possibilities how this can be done – and we will see that exploiting this information is beneficial in certain situations.

2.3 HTN and TIHTN Planning Problems

As mentioned in the beginning of the section, complexity results can be exploited in various ways. The most fundamental one is a deeper understanding of the respective problem. This is particularly important in hierarchical planning, as there is a huge variety of different formalizations. Apart from providing deeper insights, such results can also be exploited for various practical purposes, such as for (the selection of or improving) algorithms, pruning techniques, heuristics, and problem compilations. For this, we are interested in identifying the most important factors that might influence the computational complexity of hierarchical problems.

To systematically analyze hierarchical planning, we are in need of a formalism that satisfies certain requirements: First, it needs to be simplistic and second, the solution criteria should be given explicitly to make it perfectly transparent under which circumstances a partial plan is regarded a solution. We believe that simplicity is especially important for mathematical formalisms, because it makes it easier to understand the respective formalism and therefore contributes to making mistakes or misinterpretations less likely. We believe that the simplicity of the STRIPS formalization as given in Sec. 2.2 contributed significantly to its general acceptance[4]. The explicit specification of the solution criteria is especially important in hierarchical planning, because – as we will see later – there are many different interpretations and opinions about the nature and purpose of hierarchical planning. Only when it is perfectly clear under which circumstances a partial plan is regarded a solution we can state the (non-)equivalence between two problem classes, can decide whether a problem relaxation is solution-preserving, or give criteria that allow pruning partial plans during search.

Hierarchical Planning Formalisms (Related Work). Hierarchical planning is centered around the idea of hierarchical refinement in form of task decomposition. For that purpose, hierarchical planning problems define a hierarchy on the available actions dividing them into two different sets, namely the already known actions (which are now also referred to as *primitive tasks*) and *abstract* (or, synonymously: *compound*) *tasks*. Abstract tasks are considered not directly executable. Instead, they are compound, i.e., abstract representations of further primitive and/or abstract tasks. For that purpose, the domain model contains for each such task a set of so-called decomposition methods, which specify by which partial plan its abstract task can get refined by replacing it with the respective partial plan. We give a more formal description later on.

In contrast to classical planning problems, there is neither a standard description language for hierarchical problems, nor a generally accepted semantics. Relatedly, the term

[4] Antoine de Saint-Exupéry once wrote: *"Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."* [L96]

"hierarchical planning" sometimes refers to the *technique* deployed to solve problems and sometimes to the respective *problem class*. Already in 1996, Erol et al. [L76] stated:

> *There appears to be some general confusion about the nature and role of tasks in [hierarchical] planning. This appears largely due to the fact that [hierarchical] planning emerged, without a formal description, in implemented planning systems. [...] Many ideas introduced in [hierarchical] planning [...] were formalized only as they were adapted to STRIPS-style planning, and only within that context. Those ideas not adapted to STRIPS-style planning (such as compound tasks and task decomposition) have been dismissed as mere efficiency hacks.*

Some hierarchical planning approaches, like the hybrid planning approach by Kambhampati et al. [L68] or the angelic semantic planning approach by Marthi et al. [L48], just use the hierarchy to find solutions quicker than without hierarchy or to be able to produce abstract solutions, i.e., solutions that still contain abstract tasks. From a theoretical (i.e., computational) point of view, the problem that they solve is basically still a non-hierarchical one, as they only aim at finding an action sequence that satisfies the goal description – the hierarchy is just intended as advice to speed up the solution generation process of a classical problem.[5]

There is still an enormous amount of further hierarchical planning formalisms, which cannot all be mentioned here. The only survey for hierarchical planning that we are aware of is that of Georgievski and Aiello [L15]. We do want to mention, however, that there are various works that combine task decomposition with task insertion [11, L19, L20, L61, L68, L71, L86], i.e., in these approaches tasks can also be inserted into partial plans without decomposing an abstract task for this purpose. We will investigate this feature in more detail below.

Even though hierarchical planning can be used to speed up the solution generation process, its goal is usually not to find an action sequence *that reaches a certain state*, but to find an action sequence *that is obtained in a certain way*. The goal in hierarchical planning is hence often (depending on the chosen formalization) not given as a set of

[5] For Marthi et al.'s angelic semantics approach, the situation is slightly more complicated: the planning problem *does* go beyond a non-hierarchical problem description, as it also contains a set of abstract tasks (which they call high-level actions) that should be refined and their algorithm can only find plans that result from decomposing them. However, their definition of which action sequences are regarded a solution [L48, Def. 2] does *not* require those sequences to be a refinement of the initial abstract tasks. In their later paper, which proposes optimal algorithms for solving such problems [L44], their problem definition (right before Def. 1) does not contain initial abstract tasks anymore (i.e., it is, apart from the definition of the abstract tasks in the domain model, a non-hierarchical problem). Instead, the initial abstract tasks are directly handed to the search algorithm without formally being part of the problem that is solved. That is, they do not exploit the hierarchy to restrict the set of solutions [A10, A20], but to find solutions faster as well as to be able to generate abstract solutions, in which not every task in a solution needs to be primitive.

state variables that should hold after executing a plan, but as a set of (abstract) tasks that should be *achieved*. This is also pointed out by Ghallab et al. [L56]:

> *[Hierarchical] planners differ from classical planners in what they plan for and how they plan for it. In [a hierarchical] planner, the objective is not to achieve a set of goals but instead to perform some set of tasks.*

So, the definition of a hierarchical planning problem must include an initial partial plan that needs to be refined (where the latter requirement is captured by the solution criteria). Erol et al. developed a formalization of this, called *hierarchical task network (HTN) planning*, and studied its computational complexity along several dimensions [L75, L76]. In their formalization, partial plans (which are referred to as *task networks* in HTN planning – and from now on within this context), solutions also have to be a refinement of an initial partial plan/task network. However, they did not explicitly state so in terms of *declarative* solution criteria stating under which circumstances a task network is regarded a solution. Instead, they define the set of solutions via giving a recursive function that computes it [L76, Sec. 2.4]. That function basically mimics typical hierarchical planning procedures, since it computes the set of all refinements of the initial task network and extracts the set of executable action sequences from the set of task networks that only contain primitive tasks. Erol et al.'s formalism is quite expressive, allowing a boolean formula for each task network in which various constraints can be specified, such as ordering constraints, variable bindings, and constraints demanding that a state feature holds directly before or after a task, or for the complete sequence of states between two tasks (the latter constraint can be used to simulate causal links, but is more general). In this detail, Erol et al.'s formalism is closely related to the HTN formalism described by Ghallab et al. [L56, Sec. 11.5], which also allows to express these constraints. While these formalisms are quite expressive and therefore useful from a practical point of view, they are also quite technical, which makes it hard getting familiar with them. In contrast to Erol et al.'s formalism, Ghallab et al. give the criteria under which a task network is regarded a solution declaratively, requiring that there exists a sequence of decompositions that transforms the initial task network into one that is executable [L56, Sec. 11.5.3, Def. 11.12][6].

One of the more recent developments in hierarchical planning is a new formalism called *hierarchical goal network (HGN) planning* [L17, L29]. Rather than being concerned with the decomposition of tasks, that formalism is concerned with the decomposition of goals. That is, rather than using *task networks* as the basic structure, HGN planning relies on *goal networks*, i.e., partially ordered sets of goals. Consequently, decomposition methods do not state how to decompose abstract tasks, but goals. Later, HGN planning was further extended to include both goal and task decomposition; the resulting formalism is called *goal-task network (GTN) planning* [L8].

[6] Such explicit definitions were also given before, e.g., by Lotem et al. [L67].

A Novel Formalization for Hierarchical Planning Problems. We have proposed a novel formalization for hierarchical planning problems [12], for which we have basically stripped away many of the features available in Erol et al.'s formalization. The resulting formalism is closely related to the *simple task network (STN)* formalism by Ghallab et al. [L56, Sec. 11.2], since here task networks are also simply partially ordered sets of tasks – without the possibility of expressing constraints other than ordering constraints[7]. Our formalization is further distinguished from both by presenting it in a fully propositional way (but extensions to a lifted presentation were introduced later [A12, A13]). In contrast to Erol et al.'s formalization [L76], and in accordance to the one by Ghallab et al. [L56] and Lotem et al. [L67], we state the criteria under which a task network is regarded a solution in a declarative way.

The two main strengths of our formalism are as follows: it requires only a few easy-to-grasp definitions while it is still mathematically concise, and it uses declarative solution criteria that transparently define the circumstances under which a task network is regarded a solution. We refer to our formalism as HTN planning formalism [12]. When extending the solution criteria to allow inserting tasks, the respective formalism is referred to as TIHTN planning[8] (HTN planning with task insertion). The formalization presented here is just slightly adopted compared to the published variant. We also do not give the full formalism, but only its most important definitions.

The most fundamental concept is that of task networks, which are simply partially ordered sets of tasks [12, Def. 1].

Definition 2.3.1. *A* task network $tn = (T, \prec, \alpha)$ *is a 3-tuple, where:*

- T *is a finite set of symbols, so-called task task identifiers,*
- $\prec \subseteq T \times T$ *is a strict partial order on* T*, and*
- $\alpha : T \to N$*, where* N *is a finite set of task names, is called a task instance mapping.*

The set of all task networks using only names in N *are referred to by* TN_N*.*

In this definition, the partial order $\prec$ is defined on a set of so-called task identifier symbols T, which are just arbitrary symbols used for the purpose of differentiating multiple occurrences of the same task. Using the mapping α, we get the actual task given a task identifier. More precisely, α maps to the *name* of the respective task, which is part of the domain model. In case the respective name $\alpha(t)$, for $t \in T$, is a primitive task name, then we get its action by relying on a further mapping δ, which is part of the domain model.

[7] We can also express variable binding constraints in case of a lifted formalism [A12, A13].

[8] Please note that the *abbreviation* "TIHTN planning" was introduced later in the paper by Alford et al. [L18]. In our first paper on that problem class, we were referring to it as "HTN planning with task insertion" or, synonymously, as "hybrid planning" [12]. However, after that paper we used the term "hybrid planning" as done in this thesis: it refers to the fusion of HTN/TIHTN planning with concepts from POCL planning.

Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are defined to be *isomorphic*, written $tn \cong'$, if and only if there is a bijection $\sigma : T \to T'$, such that for all $t, t' \in T$ it holds that $(t, t') \in \prec$ if and only if $(\sigma(t), \sigma(t')) \in \prec'$ and $\alpha(t) = \alpha'(\sigma(t))$.

The domain model is given as follows [12, Def. 2].

Definition 2.3.2. *A* planning domain *is a 5-tuple* $\mathcal{D} = (V, N_C, N_P, \delta, M)$*, where:*

- V *is a finite set of propositional* state variables*,*
- *we require* $N_C \cap N_P = \emptyset$ *and define* $N := N_C \cup N_P$*, where:*
    - N_C *is a finite set of* compound task names*,*
    - N_P *is a finite set of* primitive task names*,*
- $\delta : N_P \to A$ *is the* task name mapping *function,* A *is a finite set of* actions*, and*
- $M \subseteq N_C \times TN_{N_C \cup N_P}$ *is a finite set of* decomposition methods

An HTN planning problem can now be defined as follows [12, Def. 2].

Definition 2.3.3. *A* planning problem *is a 3-tuple* $\mathcal{P} = (\mathcal{D}, s_{\mathrm{I}}, tn_{\mathrm{I}})$*, where:*

- $\mathcal{D}$ *is the* planning domain*,*
- $s_{\mathrm{I}} \in 2^V$ *is the* initial state*, and*
- $tn_{\mathrm{I}} \subseteq TN_{N_C \cup N_P}$ *is the* initial task network

As stated before, the solution to a task network is one that is executable and that is obtained via decomposition from the initial task network. Thus, we first need to define under which circumstances one task network is the successor of another given the application of a decomposition method [12, Def. 3].

Definition 2.3.4. *A decomposition method* $m = (c, tn_m) \in M$ *decomposes a task network* $tn_1 = (T_1, \prec_1, \alpha_1)$ *into* $tn_2 = (T_2, \prec_2, \alpha_2)$ *by replacing label* $t \in T_1$*, written* $tn_1 \xrightarrow[t,m]{} tn_2$*, if and only if* $t \in T_1$*,* $\alpha_1(t) = c$*, and there exists a task network* $tn' = (T', \prec', \alpha')$ *with* $tn' \cong_m$ *and* $T' \cap T_1 = \emptyset$*, and*

$$tn_2 = (T_2, \prec_1 \cup \prec' \cup \prec_X, \alpha_1 \cup \alpha')|_{T_2} \textit{ with}$$

$$\begin{aligned} T_2 &:= (T_1 \setminus \{t\}) \cup T' \\ \prec_X &:= \{(t_1, t_2) \mid (t_1, t) \in \prec_1, t_2 \in T'\} \cup \\ &\ \{(t_1, t_2) \mid (t, t_2) \in \prec_1, t_1 \in T'\} \end{aligned}$$

We write $tn_1 \to_{TD}^{*} tn_2$*, if* tn_2 *can be decomposed from* tn_1 *by using an arbitrary number of method applications.*

The definition simply requires that the decomposed task t is being removed from the task network and in its stead the tasks and their orderings of the method's task network tn_m are added to the resulting network. Further, all orderings that were incorporating t

prior to decomposing it are "inherited down" to *all* tasks that get added. Note that the operator $|_{T_2}$ is defined as a restriction of the respective sets to elements in T_2 thereby removing all elements that reference the decomposed task t.

Having defined the essential means to modify task networks in hierarchical planning would already allow us to define the criteria under which a task network is considered a solution. However, one of the major contributions – and in fact our main goal – of our paper was to investigate the theoretical impact of being allowed to insert tasks into task networks [12]. For that reason, we also need to define the circumstances under which one task network is the successor of another given the insertion of a task [12, Def. 4].

Definition 2.3.5. *Given a task network $tn_1 = (T_1, \prec_1, \alpha_1)$ and a primitive task name p, then tn_2 can be obtained from tn_1 by insertion of the task name p, if and only if $tn_2 = (T_1 \cup \{t\}, \prec_1, \alpha_1 \cup \{(t, p)\})$ for some $t \notin T_1$.*

We write $tn_1 \rightarrow^*_{TI} tn_2$ if tn_2 can be obtained from tn_1 via the insertion of an arbitrary number of task insertions.

In hierarchical planning, a solution must satisfy two criteria: it needs to be executable and it needs to be a refinement of the initial partial plan. We now formally define both requirements.

Definition 2.3.6. *A task network $tn = (T, \prec, \alpha)$ is called* executable *in a state $s_0 \in 2^V$ if and only if it is primitive (i.e., $tn \in TN_{N_P}$) and there exists a linearization of its tasks $t_1, \ldots, t_n$, $n = |T|$ that is compatible with $\prec$ and there is a sequence of states $s_0, \ldots, s_n$, such that $\gamma(s_i, \delta(\alpha(t_{i+1}))) = s_{i+1}$ for all $0 \leq i < n$.*

Note that according to this definition, a task network is regarded executable if there *exists* an executable linearization of its actions. We decided to use this definition, since it is the one being used by Erol et al. for their formalization [L76]. Also, it further simplified the formalization, because we do not have to define ordering insertions that way. In hybrid planning (see next section), in contrast, *every* linearization needs to be executable – we will discuss this later on.

All that is left to obtain the solution criteria is defining under which circumstances a task network is regarded a refinement of the initial task network.

Definition 2.3.7. *A task network tn_S is a solution to a planning problem $\mathcal{P}$ if and only if tn_S is executable in s_I and one of the following holds:*

- *either $tn_\mathrm{I} \rightarrow^*_{TD} tn_S$*
- *or there is a task network tn', such that $tn_\mathrm{I} \rightarrow^*_{TD} tn' \rightarrow^*_{TI} tn_S$*

In the first case, tn_S is called an HTN solution *and an* TIHTN solution *in the second.*

Note that we have two different notions of a solution, one that only allows decomposition as a means to modify task networks and one that also allows task insertions. In both cases, the syntactical representation of the domain and problem is identical – only the solution criteria are different. Still, we refer to a planning problem according to Def. 2.3.3 as *HTN planning problem* if we are interested in obtaining HTN solutions, and we refer to it as *TIHTN planning problem* if we are interested in obtaining TIHTN solutions.

As noted in the beginning of this section, the proposed formalism was used or adopted as the basis for several further theoretical results. We will give some details below (most notably about our main result on the influence of task insertion). Apart from these theoretical investigations, it was also influencing some other works in the literature [L11, L15, L16].

Complexity Results. HTN planning is generally known to be more expressive than typical classical planning – e.g., in the form of STRIPS as presented in Sec. 2.2. This was shown by Erol et al. by encoding the undecidable problem whether two context-free grammars produce a common word [L92, Thm. 8.10] as an HTN planning problem [L76]. It is easy to see that this can be achieved canonically by relying on a hierarchical planning problem via exploiting the close relationship between non-terminal symbols and compound tasks as well as between grammar rules and decomposition methods. Due to Erol et al.'s encoding with the property that the respective HTN planning problem has a solution if and only if the given (but arbitrary) context-free grammars produce a common word, the undecidability of HTN planning was shown. To show that our simplification of Erol et al.'s formalism is still expressive enough to encode undecidable problems, we replicated their proof in our formalism [12, Thm. 1] – thereby confirming the undecidability result for our formalization of HTN planning.

We give an example to illustrate their proof graphically, in particular to use the example for demonstrating the impact of task insertion. Let us consider the two context-free grammars $G = (\{H, Q\}, \{a, b\}, R, H)$ and $G' = (\{H', Q'\}, \{a, b\}, R', H')$. H, Q, H', and Q' are the non-terminal symbols of G and G', respectively. They use the same terminal-symbols a and b. The start symbol of G is H and that of G' is H'. The grammars' production rules are given by R and R':

Production rules R:	$H \to aQb$	$Q \to aQ \mid bQ \mid a \mid b$
Production rules R':	$H' \to aQ'H' \mid ab$	$Q' \to a \mid b$

The grammars' languages, i.e., the set of words produced by them, are $L(G) = a(a|b)^+b$ and $L(G') = (a(a|b))^*ab$, respectively.

In the proof that reduces the language intersection problem of two context-free grammars to the HTN plan existence problem [12, L76], the initial task network contains the start symbols of the two grammars, which are used as compound task names in the respective

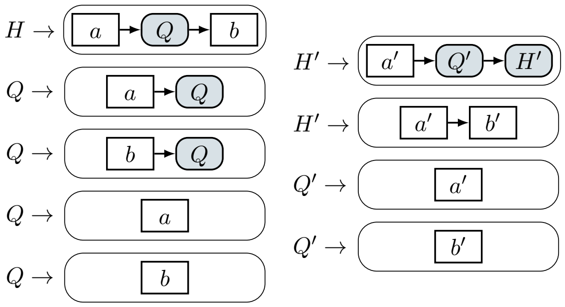


Figure 2.2: The figure shows the graphical illustration of the decomposition methods translated from the grammars' production rules. Blue boxes with rounded corners are compound task names, and white boxes with angled corners are primitive task names. (The source code of this graphic was created by both Thomas Geier and the thesis' author, Pascal Bercher.)

planning problem. All production rules are directly translated to HTN planning in the canonical way by introducing one totally ordered decomposition method per rule, and each symbol is replaced by either a compound or a primitive task name (see Fig. 2.2). Note that we can assume that the two grammars use different non-terminal symbols (otherwise: rename). Also, we can assume that they have at least one terminal symbol in common (otherwise: the answer to the problem is no, except if both grammars produce the empty word, which can be tested separately). Since we require different preconditions and effects for a primitive task encoding a terminal σ stemming from G and the same terminal σ stemming from G', we use two different primitive tasks with names σ and σ', respectively. We do not want to reproduce the full proof here, so we also do not give details how exactly the preconditions and effects look like. It suffices to mention that they ensure that a primitive task network is executable if and only if it contains two identical primitive task name sequences.

We now take a look at the example search space depicted in Fig. 2.3 to discuss the impact of task insertion. We can see that the task network tn_6 is an HTN solution, because it was produced by only using decompositions and because it contains two primitive task name sequences that encode the same word. However, we can also see that using the TIHTN solution criterion creates solutions that do not form witnesses for the intended language intersection problem. When looking at the non-solution task network tn_8, we can observe that it can be turned into the task networks tn_9 and tn_{10}. The task network tn_9 is equivalent to tn_6, so it forms a witness. However, the task network tn_{10} is executable as well (and hence forms an TIHTN solution), but it is no HTN solution, because no sequence of decomposition methods can introduce the task sequence $a'a'b'$ (because $aab \notin L(G')$). Thus, this task network is a false witness for positively answering the language intersection problem. So, what we can observe from this example is that task insertion is a powerful capability to achieve executability (as,

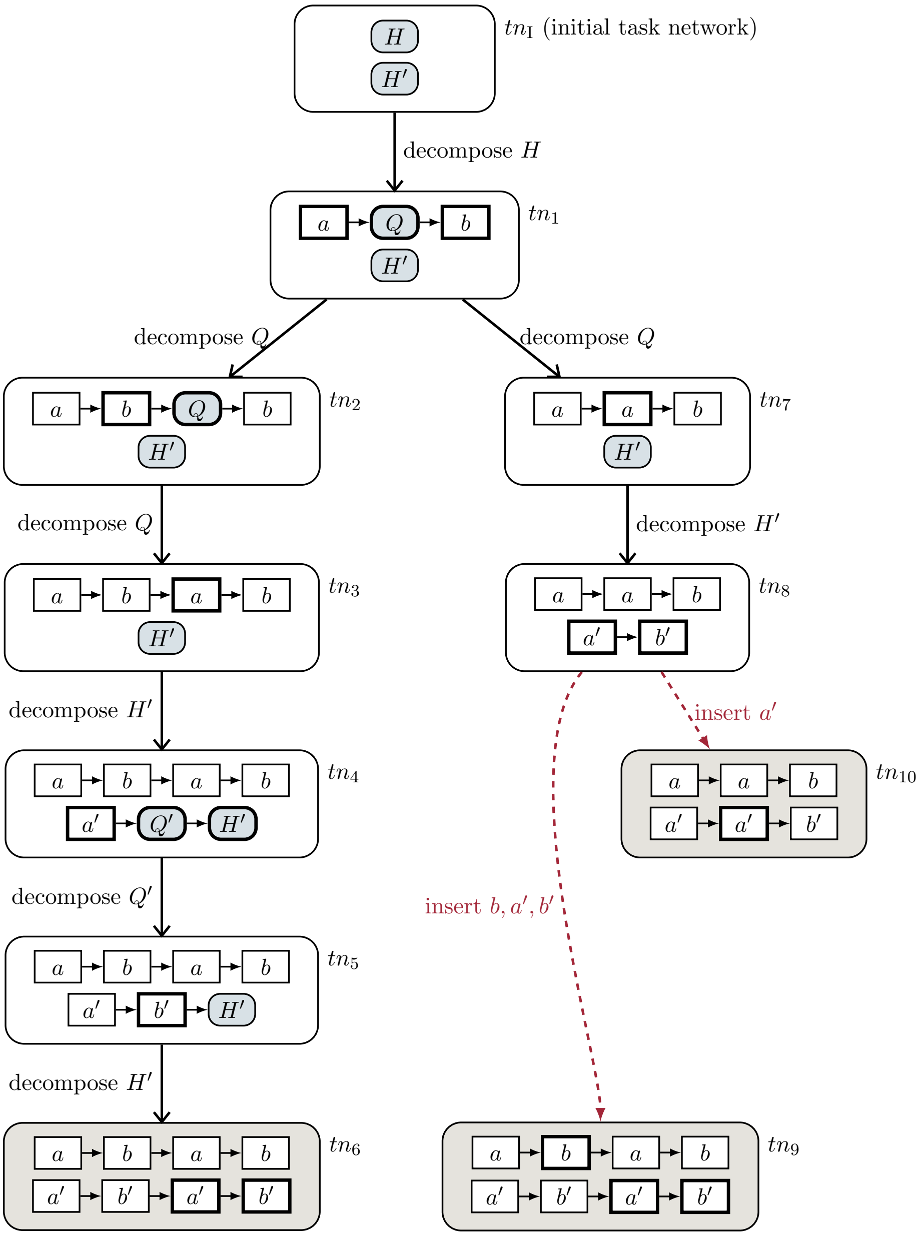


Figure 2.3: The tree depicts a fragment of a search space for the planning problem $\mathcal{P}$ with the initial task network tn_I. Primitive tasks are shown in white, compound tasks in blue. Newly introduced tasks are marked by a thick border. Task networks with gray background color are either HTN or TIHTN solutions. (The source code of this graphic was created by both Thomas Geier and the thesis' author, Pascal Bercher.)

e.g., it can turn tn_8 into a solution), but care must be spent what this *means* with respect to the set of intended solutions. In fact, we have shown that the influence of task insertion is so severe to make the resulting formalism decidable [12, Cor. 2], which means that the TIHTN planning formalism is – in contrast to the HTN planning formalism – not expressive enough to express the intended language intersection problem.

The reason for decidability is that there is no necessity anymore to use cyclic decompositions. Instead of applying the same method several times to introduce the required tasks in HTN planning, we can also apply it only *once* in TIHTN planning and introduce the remaining actions via task insertion. For instance, in our example shown in Fig. 2.3, we need to decompose Q and H' twice each to obtain tn_6, but relying on task insertion, we can also decompose them just once, leading to tn_8, which can then be turned into $tn_9 \cong tn_{10}$ via task insertion. The key observation here is that, due to the undecidability of HTN planning, there cannot be a known bound on the number of required decompositions, but we can compute such a bound for the number of additionally inserted actions, because at some point inserting actions does produce a cycle in the state space.

Complexity Results (Related Work). With the previous result, we laid the groundwork of several further investigations of task insertion – both by us and by others. In our first work concerning TIHTN planning, we "only" showed $\mathbb{EXPSPACE}$ membership for TIHTN problems [12, Cor. 1][9], since our main intention was providing insights into a completely new way to obtain decidability in hierarchical planning. Later, tight complexity bounds were provided for both totally ordered problems (where the initial task network as well as the task networks in all decomposition methods are totally ordered) and partially-ordered ones. For both, Alford et al. gave complexity results for the propositional formalization as described above, and for a lifted representation, where the model relies on a quantifier-free first-order predicate logic [A13]. We do not give any further details here, as it is outside the main scope of the thesis. Task insertion was also studied by Alford et al. who proved that TIHTN problems become solvable in $\mathbb{P}$ when being delete-relaxed and having no negative preconditions (while delete-relaxed HTN problems are $\mathbb{NP}$-complete) [L18, L21]. Recently, TIHTN problems were further studied by Xiao et al. [L7]. They showed that extending TIHTN planning by *state constraints* does not make the plan existence problem harder (or easier). State constraints were one of the constraints present in Erol et al.'s formalism, which we have stripped away (see above).

Based on our formalization of HTN (and TIHTN) planning, Alford et al. also proved a series of tight complexity results for HTN planning. As done in the analog paper that investigates the TIHTN planning complexity [A13], the respective paper about the HTN bounds [A12] bases its analysis on a lifted representation and investigates the

[9] Although our proof only states $\mathbb{EXPSPACE}$ membership, it requires only minimal adjustments to show the (presumably) lower bound of $\mathbb{NEXPTIME}$ membership.

impact of variables. Further, the impact of structural restrictions is investigated, such as considering partially and totally ordered problems as well as further restrictions on the hierarchy, such as acyclicity, to name just one example. Most of the these investigated restrictions were identified by Alford et al. [L21, L26]. For one of them, tail-recursive problems (which are problems, where recursion can only occur through the very last task in any task network)[10], Alford et al. developed a technique that translates these problems into classical planning [A7]. Tail-recursion limits the impact of cyclic method structures enough to show that there is a maximum size of any task network that can possibly be generated when performing forward progression search [L26]. Alford et al. showed how this bound can be approximated in $\mathbb{P}$ to be exploited for an automatic translation into classical planning [A7]. Details are out of the scope of this thesis, however.

Alford et al. further studied the computational complexity of both HGN problems and their extension to GTN problems [L8]. In their analysis, they addressed both task insertion as well as *task sharing*, which allows two tasks to be "fused together" given they are identical and totally unordered with respect to each other. Task sharing is a feature that can be expressed in the *Action Notation Markup Language (ANML)*, which is a language for describing temporal planning problems with hierarchical aspects [L45]. In their investigation, Alford et al. gave solution-preserving compilations and complexity results for the plan existence problem for many of the resulting problem classes.

Further related work that is based upon our formalization is concerned with deciding the *plan verification* problem. It is concerned with deciding whether a given task network is a solution for a given HTN planning problem. This problem is much more complicated than in non-hierarchical planning, because not only is the verification of a task network's executability harder than that of a totally ordered action sequence, but additionally we need to check whether such a network can be obtained from the initial task network by applying the available decomposition methods. Behnke et al. [L14] showed that this test is $\mathbb{NP}$-complete under several restrictions (e.g., when also an executable witness of the task network's executability is given, which is by itself $\mathbb{NP}$-complete to find [L76, L84]). This result was exploited by encoding this problem in a SAT problem thereby obtaining the very-first validator for HTN planning [L2]. Behnke et al. further used the theoretical HTN plan verification results as the basis for investigating the computational complexity of *change requests* in hierarchical mixed-initiative planning (MIP) [A8]. Here, a human user is incorporated in the plan generation process and he or she can request changes to the current task network presented to him or her. The results obtained help to develop MIP systems that can address such requests. This is outside of the scope of this thesis, however.

The last presented direction of related work within hierarchical planning is concerned

[10] Please note that the name *tail-recursive problem* was introduced in a later paper [A12], and not in the one where this class has originally been described [L26]. Here, tail-recursive problems do not have a name yet and are defined by the formal criterion that they must be $\leq_r$-stratifiable.

with a *direct* investigation of a planning problem's expressivity. The most common way to investigate the expressivity of a planning problem is via the investigation of the complexity of its plan existence problem. Using this indirect way of measurement ignores certain aspects of a problem's complexity, however. In particular, the plan existence problem is only interested in the computational hardness of deciding whether there *exists* a solution. However, two problems could have the same computational complexity, whereas their sets of solutions can still differ – and we argue that two planning problems can be regarded equivalent if and only if they posses the very same set of solutions (formally: there needs to be an isomorphism between the two sets). Therefore, we investigated the relationship between different planning formalisms (based on our formalization) and to the Chomsky Hierarchy [A10, A20]. Details are out of the scope of this thesis, however.

Finally, we used our novel formalization as a basis for a novel formalization for hybrid planning. Thereby, we were able to transfer many results for HTN planning to the hybrid setting, as explained next [4].

2.4 Hybrid Planning Problems

When planning technology is applied in practice, we are facing many new problems and interesting questions. For instance, one of the problems that has attracted only very little attention in the planning community is the issue of modeling support. Even if we have carefully picked our planning formalism of choice and are fully aware of its theoretical properties, it is still a tedious and, more importantly, error-prone task to model the actual problem at hand. We are thus interested in the question whether automated modeling support can be provided to prevent making modeling mistakes.

Non-surprisingly at this point, using the hybrid planning formalism as a basis is central to our answer to that question. Hybrid planning problems fuse HTN problems with POCL problems in the way that abstract tasks contain preconditions and effects. Further, the partial plans used in the model's decomposition methods may use causal links. Both features are being used to ensure that the model adheres the modeler's intent. However, hybrid problems thereby differ both from HTN as well as from POCL problems (both in terms of the syntactical problem representation, and in terms of the solution criteria), so it is not clear which theoretical properties this problem class has. As argued before, knowing about such foundational properties is not just an essential first step, but it can be exploited in many ways, so we are interested whether the results for HTN planning can be transferred to the hybrid setting.

Hybrid Planning Formalism (Related Work). By *hybrid planning* [4, 11], we refer to a hierarchical planning problem class that fuses HTN planning problems with POCL planning problems. The *name* "hybrid planning" goes back to Kambhampati et al. [L68] and Biundo and Schattenberg [L61], but related ideas have been promoted by several research groups [4]. In particular, many hierarchical planning formalisms use a formalization of abstract tasks in which they have preconditions and effects [11, L19, L20, L44, L48, L61, L62, L68, L79, L85, L86, L87, L90] – this is in contrast to the HTN formalization by Erol et al. [L76] or us [12], for which most theoretical investigations have been done.

These preconditions and effects are exploited in different ways, e.g., in combination with task insertion to be able to insert also abstract tasks into partial plans [11, L68]. Marthi et al. exploit them to describe the set of states in which an abstract task is applicable, as well as to describe the set of states reachable by *some* refinement of that abstract task [L44, L48]. This is in contrast to the semantics of preconditions and effects in our hybrid formalism, in which they can be used more flexible, but also less restrictive with regard to the set of states reachable [4]. In hybrid planning, as well as in other formalisms, in which abstract tasks use preconditions and effects, they are used to clearly state the circumstances under which a decomposition method is regarded a legal implementation of its abstract task [L61, L86, L90]. These so-called *legality* or *implementation criteria* can be exploited to prevent modeling mistakes, because they can be automatically tested and relate the preconditions and effects of the abstract task to those of the tasks used in its decomposition methods [4]. These criteria pose additional restrictions on the model, so excluding non-legal methods can, potentially, influence the computational complexity of these models. A systematic investigation of this question was not possible up to this point, however, because of the large amount of different formalizations. To systematically analyze the impact of these criteria, we hence need a unified formalism in which all these criteria can be expressed.

A Novel Formalization for Hybrid Planning Problems. To obtain a hybrid planning formalism that is well-suited for the investigation of computational aspects – under consideration of the legality criteria from the literature [L61, L86, L90] – we have adapted the previously introduced formalization of HTN planning [12] by the necessary POCL concepts [4]. Again, we do not give all the technical details here, but just the most important concepts. Further, we again slightly adopted the formalization for presentational purposes.

We start giving the definition of partial plans [4, Def. 1], which are similar to those known from POCL planning, but they may now also contain abstract tasks. It extends Def. 2.3.1 by a set of causal links. Thus, partial plans and task networks coincide given the set of causal links is empty.

Definition 2.4.1. *A* partial plan *P is a 4-tuple $(PS, CL, \prec, \alpha)$, where:*

- *PS is a finite set of symbols, so-called plan steps,*
- *$CL \subseteq PS \times V \times PS$ is a set of causal links. If $(ps, v, ps') \in CL$, then $v \in prec^+(ps')$ and $v \in eff^+(ps)$ or $v \in prec^-(ps')$ and $v \in eff^-(ps)$. We also require that every precondition variable of all plan steps is protected by at most one causal link,*
- *$\prec \subseteq PS \times PS$ is a strict partial order. If $(ps, v, ps') \in CL$ with $\alpha(ps)$ and $\alpha(ps')$ being primitive, then $(ps, ps') \in \prec$,*
- *$\alpha : PS \to N$ labels every plan step with its task name.*

The set of all partial plans using only names in N are referred to by $\mathcal{P}_N$.

The domain model of a hybrid problem [4, Def. 2] is almost identical to those of HTN problems, but the decomposition methods rely on partial plans rather than on task networks. Further, the definition of δ, which maps task names to their preconditions and effects is now defined for *all* task names – not just the primitive ones.

Definition 2.4.2. *A* hybrid planning domain *is a 5-tuple $\mathcal{D} = (V, N_C, N_P, \delta, M)$, where:*

- *V is a finite set of* state variables*,*
- *we require $N_C \cap N_P = \emptyset$ and define $N := N_C \cup N_P$, where:*
    - *N_C is a finite set of* compound task names*,*
    - *N_P is a finite set of* primitive task names*,*
- *$\delta : N \to A \cup A_C$ is the* task name mapping *function, A is a finite set of* actions*, $A_C \subseteq (2^V)^4$ describes the compound tasks' preconditions and effects, and*
- *$M \subseteq N_C \times \mathcal{P}_{N \setminus \{init, goal\}}$ is a finite set of* (decomposition) methods*.*

As noted before, we only want to include decomposition methods that adhere the modeler's intent. This can be achieved by defining criteria that must hold for the model's methods – otherwise the respective domain is not regarded valid and hence rejected. We do not give these criteria here, but discuss them in the next part (after having introduced the formalism) that is concerned with modeling assistance.

The definition of a planning problem [4, Def. 2] does, as it is the case for POCL problems, extend the domain by an initial partial plan.

Definition 2.4.3. *A* hybrid planning problem *is a pair $\mathcal{P} = (\mathcal{D}, P_I)$, where:*

- *$\mathcal{D}$ is the planning domain and*
- *$P_I \in \mathcal{P}_N$ is the* initial partial plan*.*

As it is also done for POCL planning problems, the initial state and goal description are specified in terms of two special actions encoding the initial state s_0 and the goal description g. Thus, in contrast to HTN or TIHTN problems, hybrid problems also specify a goal description. However, this is merely a formalization choice (motivated by practical considerations), because adding a goal description does not influence expressivity in any way. It can easily be expressed relying on the hierarchy [12].

We have now given all the syntactic definitions that differentiate hybrid problems from typical HTN/TIHTN problems. It remains to give the semantic changes, i.e., the solution criteria. Due to the more expressive syntax (compound tasks have preconditions and effects, and the model may contain causal links in the decomposition methods), both the definition of refinement needs to be adapted as well as the definition of executability. In case of refinement, one needs to clearly define in which way causal links that involve a compound task are inherited down upon decomposition of that task. This is a quiet technical question that has often be ignored in other formalisms or just been mentioned in informal text. A clear mathematical definition is important, however, to answer questions related to the computational complexity. We do not give a technical definition here, however, but only state informally that every causal link involving a compound task is inherited to each compatible sub task in the method's partial plan, branching over the different possibilities [4, Def. 3]. The definition of executability is taken from POCL problems: there are no open preconditions and no causal threats. Thus, in contrast to HTN planning, it does not suffice that there *exists* an executable linearization, but *all* linearizations need to be executable [4, Def. 6].

Providing Modeling Assistance. While many hierarchical planning approaches use preconditions and effects for abstract tasks [11, L19, L20, L44, L48, L61, L62, L68, L79, L85, L86, L87, L90], only a few give explicit criteria under which a partial plan is regarded a legal implementation of its abstract task [L61, L86, L90]. Without such criteria, we can basically specify arbitrary decomposition methods for any partial plan – which is a possible source of modeling mistakes: entire tasks can be forgotten, or single preconditions or effects. Even the interaction of tasks within a method can be in a different way than intended, preventing the modeled decomposition method from serving the intended purpose. As long as compound tasks are just names, it is hard to come up with an automated way to assist in finding such mistakes.

Consider the example depicted in Fig. 2.4. It shows a compound move task that is supposed to move an object (represented by the variable *?obj*) from one location (represented by the variable *?from*) to another location (represented by the variable *?to*). The task is considered not directly executable, so there is a decomposition method that states which primitive tasks implement it. Executing the two tasks *pick* and *place* after each other is intended as an implementation of the compound task. Since we use the hybrid planning formalism, we are able to specify preconditions and effects for the *move* task. Based on them, different legality criteria can ensure that the given method is fulfilling its purpose. As indicated by the lines dotted in blue, we can see that all preconditions of *move* are also used as preconditions in some tasks of the method. Further, all ef-

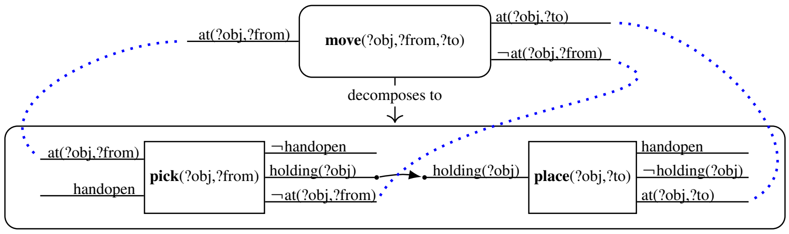


Figure 2.4: Example for a compound task (*move*) with preconditions and effects and one of its methods. The method's partial plan consists of the primitive tasks *pick* and *place* and a causal link protecting the condition *holding*. The dotted blue lines indicate how the preconditions and effects of the tasks in the method's partial plan are related to their more abstract representation. (The graphic is taken from earlier work with adjusted colors [4, Fig. 1].)

fects of *move* are used as effects of some tasks in the method. This criterion (which we called *downward compatible* [4, Def. 7]) is quite unrestrictive, but it already reduces the probability that preconditions, effects, or even tasks are forgotten to be modeled. Other criteria are more restrictive, as they take the interconnection between preconditions and effects into account. The criteria by Biundo and Schattenberg [L61], for example, would also check that the compound task's effects hold in the state that is produced after executing the method's tasks. For a deeper discussion on the investigated criteria we refer to our paper [4].

Complexity Results. Although hybrid planning is a hierarchical planning formalism, it was not clear whether Erol et al.'s and Alford et al.'s complexity results apply to it. So, it was not even clear whether hybrid planning is expressive enough to encode undecidable problems. The main reason for this is that the legality criteria restrict the set of methods that may be specified: only legal methods are allowed, so it might be that this restriction influences expressivity.

We start by investigating the computational complexity of the *plan verification problem*. We were able to transfer the recently published result for HTN planning [L14] to the hybrid setting: We showed that deciding whether a partial plan is a solution to a given hybrid problem is $\mathbb{NP}$-complete [4, Thm. 3]. This holds independent of the demanded legality criteria. The reason is that the hardness proof constructs a hybrid planning problem in which no state variables are required, so all legality criteria are automatically satisfied. Interestingly, plan verification for hybrid problems without compound tasks is in $\mathbb{P}$ [4, Lem. 3], whereas the same problem is $\mathbb{NP}$-complete in HTN planning [L14, Cor. 2]. The reason for this difference lies in the different solution criteria: for a task network being an HTN planning solution, it must *possess* an executable linearization (which is alone $\mathbb{NP}$-complete to determine [L76, L84]), but for a partial plan being a solution, *all* linearizations need to be executable, which is easier to test.

We have also investigated the *plan existence problem*. For this purpose, we have shown that, independent of which of the considered legality criteria is used, any HTN planning problem can be encoded as a hybrid planning problem in a solution-preserving manner [4, Thm. 1]. The first step in the proof is not to use any preconditions and effects for the compound tasks and not to use any causal links – then, hybrid planning domains *syntactically* coincide with HTN planning domains. However, we still need to ensure that all legality criteria hold, which does not follow from the fact that compound tasks do not have preconditions and effects. For instance, the criterion by Yang requires, related to the concept of causal links, that no precondition of a method's partial plan is threatened by another effect [L90, p. 14]. We can ensure that no criterion is violated by simply "moving" each primitive task in a method on its own and introducing an effect- and precondition-free compound task that is associated with exactly this method. That way, each partial plan contains either only compound tasks or just a single task. In both cases, all legality criteria hold. However, due to the different solution criteria, there is no 1-to-1 correspondence between the set of solutions of the given HTN problem and the hybrid problem that encodes it. Instead, for each executable action sequence in any HTN solution task network there exists a solution plan containing the same sequence [4, Thm. 1]. From this we can conclude that hybrid planning is as expressive as HTN planning in the general case, i.e., semi-decidable and undecidable [4, Thms. 4 and 5]. We can also conclude that many structural special cases of hybrid planning (such as totally ordered problems) are as hard as in HTN planning (namely those structures that are not affected by the encoding; see the paper for details).

With this analysis, we have provided the first theoretical investigation of hybrid planning. At the time being, we did only investigate the purely hierarchical solution criteria. Considering hybrid planning with task insertion is future work.

There is more to life than forward search
using primitive grounded actions.

Stuart Russell, Invited Talk at ICAPS 2015

3 Search and Heuristics

Motivation. Hierarchical planning is computationally hard (see Sec. 2). Both HTN planning as well as hybrid planning is in the general case undecidable. Even though this worst-case complexity does not always occur in practice, problems are ordinarily still too complex to solve them with blind, uninformed algorithms. So, we are in need of an *informed* algorithm that is able to solve hybrid problems and all its sub classes mentioned in the last section heuristically. Solving planning problems as quickly as possible is especially important if the respective system is applied for our endeavor of assisting human users, since we want to reduce waiting times as much as possible.

The algorithm should further be able to generate *optimal* solutions, which is of particular importance for assisting human users. Sup-optimal planners are often quite good in solving problems quickly, but the solutions they produce might be counter-intuitive to a human user in case they contain obvious redundancies (which often happens in practice), such as plugging in a cable, then unplugging it again, followed by plugging it in again.

Both efficiency as well as optimality can be guaranteed by providing the search algorithm with heuristics with certain properties: they need to be well-informed to find solutions quickly, and they need to be admissible to find optimal solutions.

Summary of Core Contributions. The first contribution for solving hierarchical problems quickly is the development of a heuristic search algorithm, called Panda, for hybrid planning problems [7][1]. It can solve the full spectrum of problem classes elaborated on in the last section (cf. Fig. 2.1), i.e., hybrid planning problems without and with task insertion, HTN and TIHTN problems, as well as POCL and POP problems. In its core, the algorithm is a standard POCL planning procedure – but extended in a way that allows to cope with abstract tasks. As such, it performs search in the space of partially ordered partial plans, which rely on causal links to ensure applicability of its tasks. The respective planner is one of the very few domain-independent heuristic search planners for hierarchical planning problems. Its performance is evaluated empirically and compared to standard search strategies from the literature in several benchmark domains.

[1] The proposed algorithm is an improvement of an earlier version of the respective system [L38].

As a basis for informed heuristics, we exploit the so-called *Task Decomposition Graph (TDG)* [1, 10], which represents the AND/OR structure underlying any hierarchical planning problem. We introduced the concept of so-called *local hierarchical landmarks* of an abstract task t, which are all tasks (primitive or abstract) that occur in any sequence of decompositions leading from a partial plan containing t to a solution. These local landmarks are used as the basis for informed search strategies to provide a ranking for the usefulness of different plan refinements [10]. These landmark strategies have been particularly designed for the use in a specialized version of the Panda system, a predecessor of the one proposed in this thesis. We have adopted the core ideas behind the landmark-based ranking strategies for that particular system and transferred them to the novel hybrid planning algorithm in the form of standard heuristics that estimate the effort of turning a partial plan into a solution [7]. The heuristics are applicable to all hierarchical problem classes, i.e., independent of whether task insertion is allowed or not, and independent of whether causal links are present in the domain and problem description.

The heuristics mentioned before estimate the remaining effort to turn a partial plan into a solution – based on landmarks [7]. However, such landmarks do not always exist, even if the problem is solvable. Further, even if landmarks do exist, the extraction procedure is not always capable of identifying all of them. We have thus developed further heuristics that are also well-informed in the complete absence of any landmarks. We introduce two such heuristics, which both share the same general idea of how estimates are computed, but they are different in what kind of goal distance they estimate. They both traverse the TDG exploiting its AND/OR structure in the canonical way by minimizing over the available methods for estimating the effort of an abstract task, and by summing over the effort of the tasks in a method's partial plan for estimating the effort of such a method [1]. The first heuristic based on this idea is called *modification-aware TDG heuristic*. It estimates the least number of modifications (i.e., causal link insertions and task decompositions) the Panda planner needs to perform to turn a partial plan into a solution. The second heuristic is called *cost-aware TDG heuristic*. It estimates the costs of the actions that need to be inserted into the given partial plan to turn it into a solution. This estimate is admissible thereby allowing to find optimal solutions. In fact, this is the very first admissible heuristic for both hybrid and for HTN planning problems. The resulting planner configuration is thus the very first planner that can find optimal solutions for standard HTN and hybrid problems heuristically[2].

We have also shown that recomputing the TDG during planning can improve heuristic accuracy [1]. The key observation here is that the heuristics as proposed above are *preprocessing heuristics*, because the TDG is computed only once prior to planning. Recomputing the TDG during planning can remove parts of it that have become un-

[2] Note that there are further optimal heuristic search planners for hierarchical panning in the literature. However, they assume that admissible heuristic estimates are given in the input, or they are defined for different hierarchical problem classes, as mentioned in Sec. 3.3.

reachable due to a different action set that is reachable from the current search node. We have shown in which situations during planning rebuilding a TDG might be beneficial to avoid recomputations that would not lead to increased heuristic accuracy. Since the recomputation comes at the cost of additional runtime overhead, the recomputation does not always pay off in terms of runtime. The achieved search space reductions vary in different problem instances; the largest reductions are up to 93%, which coincide with runtime reductions of 40%.

We have also developed heuristics for the special case of hierarchical planning that arises in situations, in which all abstract tasks have already been decomposed and we end up in partial plans, in which all tasks are primitive. These are the problem classes POP and POCL – assuming that task insertion is allowed. The first heuristic that we have developed is called *SampleFF* [9]. It is based upon the well-known FF heuristic from classical planning [L63]. In contrast to the FF heuristic, which estimates the goal distance for a given current state, the SampleFF heuristic estimates the goal distance for a given current partial, non-linear plan. In contrast to other state-of-the-art heuristics that estimate the goal distance of such partial plans [L59, L64], SampleFF exploits causal links, the partial order, and the negative effects of the plan steps in the given partial plan to a large extent. To be able to do so, it samples a fixed number of linearizations of these tasks and computes, based on the FF heuristic, a delete-relaxed plan in which all these sampled tasks are contained. Due to the exploitation of all the information that is available in a given partial plan, the heuristic is very well informed. Among all evaluated heuristics, it was the only one being able to prove the unsolvability of an unsolvable problem instance – all other evaluated heuristics incurred timeouts. However, it requires more time per search node, which makes it less efficient in terms of solving time compared to the least informed, but extremely fast, POCL heuristics from the literature [L59, L64]. It does, however, show great potential for reducing the search space by pruning search nodes that cannot be turned into solutions.

We have come up with another technique for estimating the goal distance of a primitive partial plan. This technique is more general than one particular heuristic, as it makes heuristics from state-based classical planning directly applicable to POCL planning [8]. It works as follows: we encode a given partial plan for which a heuristic should be computed into the given planning problem, such that we do not rely on that partial plan anymore. Instead, every solution to the altered problem is a refinement of that partial plan. That way, we can use any heuristic from state-based planning[3] to estimate the goal distance from the initial state of the altered problem and use it as heuristic for the partial plan. Using this technique gives us the first well-informed admissible heuristics for POCL planning, since any admissible state-based heuristic serves as an admissible heuristic for POCL planning. Our empirical results were quite promising: the well-known (admissible) LM-cut heuristic for state-based planning [L36] produced

[3] Since heuristics perform problem relaxations, not *every* heuristic will be suited for our encoding, as they might relax the information that encodes the given partial plan within the new problem [8].

significantly smaller search spaces than the currently best-performing (non-admissible) heuristics for POCL planning.

The above-mentioned core contributions are published in the following publications:

[1] **P. Bercher**, G. Behnke, D. Höller, and S. Biundo. "An Admissible HTN Planning Heuristic". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017).* AAAI Press, 2017, pp. 480–488. DOI: 10.24963/ijcai.2017/68

[7] **P. Bercher**, S. Keen, and S. Biundo. "Hybrid Planning Heuristics Based on Task Decomposition Graphs". In: *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS 2014).* AAAI Press, 2014, pp. 35–43

[8] **P. Bercher**, T. Geier, and S. Biundo. "Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning". In: *Advances in Artificial Intelligence, Proceedings of the 36th German Conference on Artificial Intelligence (KI 2013).* Springer, 2013, pp. 1–12. DOI: 10.1007/978-3-642-40942-4_1

[9] **P. Bercher**, T. Geier, F. Richter, and S. Biundo. "On Delete Relaxation in Partial-Order Causal-Link Planning". In: *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013).* IEEE Computer Society, 2013, pp. 674–681. DOI: 10.1109/ICTAI.2013.105

[10] M. Elkawkagy, **P. Bercher**, B. Schattenberg, and S. Biundo. "Improving Hierarchical Planning Performance by the Use of Landmarks". In: *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012).* AAAI Press, 2012, pp. 1763–1769

3.1 Search Algorithm

In this section, we introduce our heuristic hybrid planning system Panda (Planning and Acting in a Network Decomposition Architecture) that is able to solve all problem classes that were introduced in the last section (cf. Fig. 2.1).

Panda is a planning framework that also features components for *linearizing plans* [2, A21], *repairing plans* [2, 6, 11, L41], *explaining plans* [2, 6, 11, L28], and *verifying plans* [L2, L14]. For a more detailed discussion on these user-centered planning capabilities, we refer to Sec. 4.2. Here, we focus on Panda's search routine.

Panda has a development history of approximately 15 years. Before introducing it and its history, we give a short overview of related work.

Solving Hierarchical Planning Problems (Related Work). As noted in Sec. 2.2, the currently most-pursued planning approach is solving classical planning problems via forward search in the space of states using primitive grounded actions. In hierarchical planning, there is a similar approach based on progressing a current state. The respective system is called SHOP2 [L58] – it solves a slight variation of the standard HTN problems as they were presented in Sec. 2.3. The most important difference is that the HTN problems that SHOP2 solves supports so-called *method preconditions*, which are especially designed for this progression-based planner: they specify the circumstances under which a decomposition method may be used to decompose an abstract task. These preconditions further allow to specify a precedence of the available methods and are a means to encode domain- and/or problem-specific control knowledge. Due to these powerful preconditions, SHOP2 is no heuristic search planner. Instead, it performs blind depth-first search. Its efficiency completely depends on the hand-tailored expert knowledge that is given in terms of the task hierarchy and, particularly, the method preconditions. This makes the modeling process especially complex and important, as the modeler needs to ensure that the respective control knowledge allows solving the problem quickly. The formalism presented in Sec. 2.3, upon which our planner relies, comes without such hand-tailored knowledge. Instead, the performance of any planning system solving such problems stems entirely from the search strategy and domain-independent heuristics.

The angelic semantic planning approach by Marthi et al. is able to generate "abstract solutions" – solutions that may still contain abstract tasks, but are guaranteed to have a primitive refinement that solves the given problem [L44, L48]. Their approach performs search in the space of partial plans, similar to PANDA. It is also capable of finding optimal solutions, but they rely on optimistic estimates for their abstract tasks that are given in the input. Finding such estimates in a fully automated and domain-independent way remained future work.

We do not want to give a complete overview of related hierarchical planning systems here, but only mention the ones closest related to ours (which are still in use). Instead, we mention overviews that are concerned with hierarchical planning systems. A theoretical perspective is taken by Alford et al. [L26]. They characterize HTN planning as search in a problem space and identify four kinds of these problem spaces. We have already seen two of them: search in the progression space (which SHOP2 is performing) and search in the decomposition space (which PANDA is performing). Only a few actually implemented systems are mentioned as related work in this categorization. A more practical view is taken by Georgievski and Aiello [L15]. They give an overview of some of the very first hierarchical systems, starting with the planner NOAH (Nets of Action Hierarchies) that was first described in 1975 [L93]. A more recent overview of HTN and TIHTN planning systems is given by us [1].

Apart from these "standard" planning systems to solve hierarchical problems, there is also a technique that solves such problems based on non-hierarchical planners. Alford et al. developed a compilation technique that translates a given *totally ordered* HTN problem into a solution-preserving PDDL problem [L21, L35]. Despite the decidability of that problem class [A12, L76], that translation still requires an additional parameter specifying the maximal decomposition depth – they later showed how this parameter can be computed automatically [L26]. In a recent paper, Alford et al. extended the previous translation to being able to cope with tail-recursive HTN problems [A7]. However, also non-tail-recursive problems may be translated into non-hierarchical problems. Due to the undecidability of HTN planning, this requires the translation into a (possibly infinite) sequence of non-hierarchical problems, however.

The PANDA Algorithm. The planner PANDA meanwhile exists in three different versions. Based on an initial version, PANDA$_1$ [L38], there have been two reimplementations, which also differ from PANDA$_1$ on the level of pseudo code, i.e., they are not just technical improvements. PANDA$_1$ and PANDA$_2$ are implemented in Java, PANDA$_3$ is implemented in Scala (mainly the preprocessing routines) and in Java (mainly the search routines). Both PANDA$_2$ and PANDA$_3$ conceptually work in the same way, i.e., they are identical on the level of pseudo code [7, Alg. 1]. We mention the differences of these planners to each other, and to their ancestor PANDA$_1$, below. All versions are so-called *decomposition-based planners* [L26], i.e., they perform search in the space of partial plans. They do so by relying on techniques known from POCL planning, i.e., they use causal links to ensure the executability of the partial plans and maintain only a partial order of its plan steps to follow the principle of least commitment.

The newest versions, PANDA$_2$ and PANDA$_3$, both extend the standard POCL planning procedure by Williamson and Hanks [L77] for non-hierarchical planning in such a way that it can handle abstract tasks. As such, they can handle the full spectrum of planning problems that was introduced in Sec. 2 (see also Fig. 2.1). The search procedure [7, Alg. 1] has two decision points: first, a most-promising partial plan is selected among all partial plans that were generated so far (and that were not already processed). This selection is based on a search strategy (such as *breadth first*, *depth first*, etc.) as well on heuristics (which are used by informed search strategies such as *greedy search* or A^*). For the selected partial plan, all its flaws are computed. In Sec. 2.2, we have already seen two types of flaws: *open precondition flaws* and *causal threat flaws*. Since the hybrid planning procedure extends standard POCL planning, both flaw types are also used for solving hybrid (or HTN and TIHTN) problems. In addition, we need the new flaw type *abstract task flaw*, which is raised for each abstract task in the current partial plan. Given a selected partial plan and the set of its flaws, the planner chooses one of these flaws to address it. Note that this choice point is – contrary to the selection of a partial plan – not a backtrack point. *All* flaws need to be addressed at some point and the order in which they are chosen does neither influence correctness nor completeness. It can, however, have tremendous influence on performance, so several flaw selection strategies were proposed in the literature [L74]. For the selected flaw, a set of so-called modifications is computed; they describe how the respective flaw can be addressed. The mapping which modification addresses which flaw for a hybrid planning problem is given in our paper [7].

We now briefly summarize the development history of the different PANDA versions. The full pseudo code of the initial version, PANDA$_1$, is given in Algorithm 2.2 (page 70) in the dissertation of Schattenberg [L38]. A slightly more compact description is given by Elkawkagy et al. [L33]. We later simplified the description of the algorithm significantly [10]. Note, however, that PANDA$_1$ also features scheduling capabilities, which are only reflected in its full pseudo code description [L38]. PANDA$_1$ was designed as a comprehensive experimental platform to be as easily extendable as possible – at the cost of runtime. PANDA$_2$ [7] is the first reimplementation with focus on efficiency – at the cost of being less easily extendable. The main search procedure changed in three major aspects: First, PANDA$_1$ relies on two different ranking functions; one that orders all partial plans and one that orders all modifications of all flaws of the given partial plan. Instead, PANDA$_2$ (as well as PANDA$_3$) relies on just a single search fringe as it is done by standard POCL planning procedures [L77]. Second, PANDA$_1$ addressed always *all* flaws[4] of the given partial plan [11, L33, L38], whereas PANDA$_2$ only addresses a single one [7]. That way we have eliminated an unnecessary choice point in the algorithm, thereby reducing the branching factor in the search space significantly without loosing solutions. This further enables the use of standard flaw selection strategies known for POCL planning [L74]. Third, PANDA$_1$ always computes flaws from scratch, although most flaws of a parent search node are unaffected by the performed modifications. Instead, PANDA$_2$ performs an incremental flaw computation that reuses the flaws of its parent node and removes and adds flaws that might have been resolved or that were raised as a consequence of the last applied modification. This technique is not reflected in the published pseudo code, however [7]. The search time reductions from PANDA$_1$ to PANDA$_2$ were significant. Problems that needed 15 to 20 minutes to be solved relying on informed search strategies with with PANDA$_1$ [10] can often be solved in less than one second when relying on uninformed breadth first search with PANDA$_2$ [7]. We assume that these speed improvements cannot only be attributed to eliminating a branching point, but also *significantly* to a highly optimized implementation[5]. PANDA$_3$ is a further reimplementation, which fuses the best of the two previous extremes: it is almost as fast as PANDA$_2$ while being as easily extendable as PANDA$_1$. Its search procedure PANDA$_3$ is the same as the one of PANDA$_2$ – except for the incremental flaw computation, which is at the moment only partially implemented in PANDA$_3$.

Only a few of the empirical results of this thesis were done with PANDA$_1$ [10]; most of of

[4] We want to note that the implementation of PANDA$_1$ featured an approximation of addressing just a single flaw. Let $m_1, \ldots, m_n$ be the (ordered) sequence of modifications that resulted from the modification ordering function. Then, PANDA$_1$ can remove all modifications m_i, $i > j$, given that there is a flaw, for which all modifications are contained in the sequence $m_1, \ldots, m_i$.

[5] In addition, the evaluation based upon PANDA$_2$ [7] was done 2 years after the evaluation based on PANDA$_1$ [10], so there have also been hardware improvements.

the techniques were implemented within PANDA$_2$ [7, 8, 9]; our most-recent heuristics for hybrid and HTN planning are implemented within PANDA$_3$ [1]. We are currently in the process of making PANDA$_3$ available as an open source project. The code will be made publicly available at www.uni-ulm.de/en/in/ki/panda/.

3.2 Heuristics for POP and POCL Planning Problems

The core algorithmic principles behind our hybrid planning system PANDA are those of standard POCL planning algorithms – extended by a further flaw class and further modifications to being able to cope with abstract tasks. This makes improving techniques for POCL planning problems a canonical first step to improve the overall performance of the system. In particular, POCL planning problems coincide with hybrid planning problems with task insertion, given that all abstract tasks have already been decomposed. So, applying specialized POCL heuristics to such search nodes might be more efficient than applying heuristics that are applicable to partial plans containing abstract tasks. Well-informed POCL heuristics may also serve as the basis to develop heuristics for hybrid planning problems, e.g., by computing the set of landmarks for each abstract task (see Sec. 3.3), replacing the respective abstract tasks by their primitive landmarks, and computing the heuristic based on the resulting partial POCL plan.

We want to emphasize that even apart of our main intention of improving our hybrid planning system, improving the state of the art in POCL planning is an important means of its own: namely for solving classical, non-hierarchical planning problems. Back in the late 1980's, POCL techniques have been "*known*" as the superior approach to planning [L91]; at the moment, however, their usefulness for solving classical problems is generally doubted, as bluntly pointed out by Nebel in a key note at the German AI conference 2013: *"Search in the space of incomplete, partially ordered plans is inefficient, there are no good heuristics available – and it is obsolete."*. However, the only evidence for the inefficiency of POCL techniques is the – today's – superiority of the highly optimized state-based planning approaches, for which exist a wide variety of different planning heuristics. Almost no researchers are working on POCL planning for about a decade now, which we deem one of the main reasons for its current inferiority to state-based planning techniques. Our work in that field, both theoretically (see Sec. 2.2), as well as practically [8, 9, A23], may be one of many steps in turning the current situation into the opposite.

Heuristics for POCL Planning (Related Work). A brief introduction and overview of some of the existing POCL planning systems has already been given in Sec. 2.2. Here, we focus on related work that is concerned with automatically deriving well-informed heuristics for POCL planning. Please note that, due to the large amount of work in the field of improving POCL planning systems, we are focusing on the most-successful ones that do – similar to well-informed heuristics in classical planning – perform some form of problem relaxation.

One of the most successful planners based on POCL techniques is CPT (Constraint Programming Temporal planner), which combines the flaw-based branching scheme of POCL planning with powerful pruning rules that are implemented via constraint programming techniques based upon heuristics [L52]. CPT is an optimal planner for *temporal* classical planning, i.e., it finds plans that minimize the makespan (the completion time when parallel action execution is allowed). The heuristics that are used for the constraints are based upon the admissible h^m *heuristic(s)* for classical planning [L50, L65].

The planner RePOP (Reviving Partial Order Planning) [L64] has introduced the *relax heuristic*, h_{relax}, for POCL planning. It is an adaptation of the *FF heuristic* known for classical planning [L63] to work on partial plans rather than on states. The *relax* heuristic basically extends the standard FF heuristic by an additional step: Given a partial plan and its open preconditions, it uses these open preconditions as the goal state that it passed on to the (state-based) FF heuristic, which then computes a delete-relaxed plan for it. The cost of this delete-relaxed solution serves as heuristic value for the given partial plan. The actions that are already present in the given partial plan are accounted for by excluding those preconditions that are already protected by a causal link, and by not accounting for the costs of those actions in the delete-relaxed solution, which have a non-delete-relaxed pendent in the given partial plan.

The planner VHPOP (Versatile Heuristic Partial Order Planner) [L59] uses ideas similar to those of RePOP to exploit classical planning heuristics for POCL planning. It extends the state-based *Add heuristic* [L70] to be used for partial plans. The *add* heuristic computes the heuristic estimate of an action based on its action cost plus the sum of the heuristic estimate of its precondition literals, thereby assuming subgoal independence. Each precondition literal is estimated by the heuristic estimate of its cheapest achiever (i.e., action) [L70]. The *Additive heuristic for POCL planning*, h_{add}, is defined in same way; however, similar to the *relax* heuristic, it uses the open preconditions of the given partial plan as goal description, i.e., as input for that heuristic. To account for actions that are already within the given partial plan, they propose the *Additive heuristic for POCL planning reusing actions*, h^r_{add}. This heuristic is distinguished from h_{add} by an additional filter: instead of using all open preconditions in the given partial plan as goal description, h^r_{add} uses only those for which there is no action within the plan that can, according to the partial order and the variable constraints, possibly serve as a producer. In this regard, h^r_{add} does, in contrast to h_{relax}, exploit the partial order order of the given partial plan as well as the effects of the actions that are already present to a limited extent.

The SampleFF Heuristic. We have just seen two of the very few existing heuristics for POCL planning; the *relax* and *add* heuristic. They both rely on delete relaxation and, more importantly, as explained in Sec. 2.2, they ignore parts of the information obtained during search; recall that the complete search progress is encoded in the current search node, which in POCL planning is the current partial plan. Still, both the *relax* heuristic as well as the *add* heuristics ignore a large amount of information in the given partial plan: the partial order is ignored completely (except by h^r_{add}, which exploits it by some extent – as explained above), the effects of the present actions are ignored (again: except by h^r_{add}, which uses them to "filter" out preconditions of unprotected preconditions – as explained above), and, most importantly, the pruning power of causal links is completely ignored: they are just used to "filter" out preconditions, which are already supported, but not to restrict the applicable actions[6].

In Sec. 2.2, we have seen that maintaining all this information and delete-relaxing only the actions in the domain results in an $\mathbb{NP}$-complete problem. However, it becomes trivially decidable in $\mathbb{P}$ when the partial plan is totally ordered [9]. We have developed a heuristic that exploits this result to have a tractable heuristic on the one hand while still being well-informed due to exploiting all effects of the actions present in the current partial plan and, more importantly, exploiting the present causal links' pruning power by respecting their protected conditions. In particular by exploiting the pruning power of causal links, we were aiming at more informed heuristic estimates.

The SampleFF heuristic [9] consists of three steps:

1. Uniformly sample a fixed number of linearizations.
2. For each linearization compute a delete-relaxed solution (if one exists).
3. Combine the different heuristic values. Cope with the special case that none of the linearizations admits a solution.

Step 1. The sampling part approximates the computationally hard task of finding a linearization that admits a delete-relaxed solution. We have experimented with different numbers of samples. The computation effort scales approximately linearly in the number of linearizations with the benefit of being more informed for every additional linearization and with an increased likelihood of finding linearizations that actually do admit a delete-relaxed solution. For the latter, we have also implemented a technique that reuses successful linearizations from a parent search node for the current one.

Step 2. For a given linearization, all available delete-relaxed actions are applied to the initial state and the succeeding ones until a fix point state is reached. This is essentially the same as building a mutex-free planning graph [L69], as it is done by the FF heuristic [L63]. We then check whether the first plan step in the linearization is applicable in that

[6] As explained in Sec. 2.2, each causal link (ps, φ, ps') between the plan steps ps and ps' prevents to insert any action between them that has an effect which undoes the protected condition φ.

state. It not, the linearization can be discarded – no heuristic (smaller than infinity) can be extracted for it. If it is, the linearization's first plan step's effects are applied to that state. Then, again, all available delete-relaxed actions are applied to reach another fix point. This is repeated until all actions of the given linearization have been applied and the goal description is captured. For two plan steps ps_1 and ps_2 (ps_1 preceding ps_2) that share a causal link with the (positive) protected variable v, no action with the negative effect v can be placed in between ps_1 and ps_2. Note that this also holds for delete-relaxed actions (i.e., even if the threatening delete effect has been relaxed away), because such actions can not possibly be used at the respective position in the non-relaxed partial plan. So, the more causal links are within a partial plan, the more (delete-relaxed) actions can be excluded to be used for solving the delete-relaxed problem, and the more accurate the heuristic becomes. This is in contrast to the *relax* and *add* heuristics for POCL planning, where more causal links make the problem *easier*, although it is actually becoming more restricted. So, given the respective sets of available delete-relaxed actions (which may be different for the different layers of the planning graph), a delete-relaxed solution is extracted by the FF heuristic [L63], the cost of which serves as heuristic for the given linearization.

Step 3. The different heuristic estimates can be combined in different ways. So far, we have taken the minimum. In the special case where none of the linearizations admits a delete-relaxed solution it might be that there still exists such a linearization but that was not drawn by the sampling procedure. However, it might also be the case that the partial plan is a dead-end altogether; however, this cannot be formally proved unless the number of samples is at least as high as the number of existing linearizations. So, if this is not the case we currently return the number of open conditions to prevent being uninformed in such situations.

Results. In our empirical evaluation we have compared different variants of our heuristic with the current state-of-the-art heuristics in POCL planning. Different variants are obtained from different numbers of samples and from different possibilities of which causal links are exploited. The results are giving a mixed picture. Regarding coverage (number of solved problems within a given time window), *relax* and *add* show better results than the best-performing variant of SampleFF. This can be attributed to the high computation times that SampleFF requires on the one hand and one important finding on the other: The most informed variant of *SampleFF* (which uses the highest number of generated samples and all causal links) was able to disprove the existence of a delete-relaxed solution for all linearizations in 36% of all created search nodes (being summed over all runs of all problems). This means that in these cases the respective partial plans might actually be dead-ends. Only in a few cases, however, the respective plan could be discarded – due to the non-exhaustive number of sampled linearizations. This still shows the great potential of using the pruning power of causal links to reduce the search space. Most interestingly, for one (unsolvable) problem instance, only the

SampleFF heuristic was able to prove the problem's unsolvability[7]; all other heuristics incurred timeouts.

Exploiting Classical Planning Heuristics in POCL Planning. As we have seen in Sec. 2.2, POCL problems have the same computational complexity as classical planning problems. Thus, it must be possible to compile one into the other in polynomial time. We have given such an encoding that enables us to use classical state-based planning heuristics in the context of POCL planning [8]. The transformation is an extension of the one given by Ramírez and Geffner [L37] which they developed for a totally ordered sequence of observations in the context of plan recognition. We can encode a given partial plan P within the given classical planning problem. Then, we define $h(P)$ by means of $h'(s)$ with h' being some state-based heuristic and s being the initial state of the altered problem. We do so by adding the plan steps of P as additional actions to the problem, where these additional actions extend their original preconditions and effects by means of additional state variables that ensure that these additional actions are executable exactly once and only in an order that is compatible with the one given in the respective partial plan. Further, the goal description is extended by these additional state variables to ensure that any solution contains these actions. Taken together, the encoding ensures that any solution to the new classical planning problem encodes a solution to the POCL planning problem (that is induced by the given partial plan)[8].

Care must be taken in choosing the respective state-based heuristic: Since every heuristic performs a relaxation of the given planning domain, it might be that certain heuristics work better than others – or not at all. For instance, imagine a so-called pattern database (PDB) heuristic that restricts the set of available state variables to a subset thereof [A25, A26]. If such a pattern (i.e., the restricted variable set) does not contain any of the additional state variables, the current partial plan will consequently be ignored. Other issues arise for heuristics that spend a lot of time for preprocessing (but make up for it by having better-informed heuristic values per search node), such as PDB heuristics or their generalization merge & shrink [L47], because the preprocessing has to be re-made (or adapted) for each search node due to the changed action set.

Results. For our empirical evaluation, we wanted to test a wide variety of state-based heuristics without the need to implement all of them. We thus modified the planning system Fast Downward [L51], since it features most of the known classical planning

[7] Despite the non-exhaustive number samples, proving unsolvability was possible, because the deployed search strategy favors addressing flaws that result into partial plans that only have a single linearization for as long as possible. Hence, even "sampling" just a single linearization can prove that this partial plan may not be refined into a solution. Further, as noted before, partial plans may also be discarded if all linearizations are considered, which is feasible as long as there are just a few.

[8] As already noted in Sec. 2.2, our published encoding for causal links is only fixed-parameter tractable, namely in the size of the largest precondition plus the size of the largest effect [8]. We have also developed an encoding for causal links that runs in polynomial time, but it is not yet published.

heuristics, to return the heuristic of the initial state and shut down afterwards. We then created the respective PDDL files from the given search nodes and passed them on (via a RAM disc) to the modified planner to use its heuristic for the respective partial plan. Due to the overhead of creating the PDDL files for each search node and starting Fast Downward for each of them, we were not comparing run times, but focusing on the produced search space size. Two heuristics were doing particularly well: merge & shrink [L47] and LM-cut [L36]. Although merge & shrink is doing well in terms of the produced search space, we do not expect it to scale well in terms of runtime for the reasons explained above. LM-cut, in contrast, does not rely on a time-consuming preprocessing step, so we are slightly optimistic that it works well for our approach when being natively implemented (which is currently ongoing work).

The results look quite promising: although LM-cut is an admissible heuristic and *relax* and *add* are not, it still produces smaller search space sizes, so it acts more informed [8]. We further want to emphasize that the here described technique forms the first well-informed admissible heuristic for POCL problems[9] – if the compilation is used in combination with a well-informed admissible state-based heuristic such as LM-cut and thus allows to find optimal solutions. This becomes especially important when a POCL algorithm is used for providing user assistance (see Sec. 4), since in some application domains it might be important to present an *optimal* plan to the user – rather than just *some* plan.

3.3 Heuristics for HTN and Hybrid Planning Problems

We are now considering how domain-independent heuristics for HTN and hybrid planning problems can be derived. In addition to our main intention of having well-informed heuristics for solving the respective problems as quickly as possible, we are furthermore interested in finding optimal solutions – which requires the respective heuristics to be admissible.

Heuristics for Hierarchical Planning Problems (Related Work). As mentioned in the related work part of Sec. 3.1, there are only a quite limited number of hierarchical planning systems [1, L15]. Only a few of them are standard heuristic search planners, in which search nodes are ranked relying on automatically computed domain-independent heuristics. Instead, in some approaches, no heuristics are required because search control knowledge is encoded into the model itself [L58] or because a specialized algorithm is used [L42, L67]; other approaches do rely on heuristics, but they are not concerned

[9] Please note that the temporal POCL planner CPT [L52] also relies on an admissible heuristic. It does, however, minimize the makespan instead of action costs (or their number).

with solving standard HTN planning problems, but with variations thereof, such as, for example, HGN problems [L6, L13, L25].

In contrast to the previously mentioned works, only few publications are concerned with solving standard hierarchical problems (such as HTN and hybrid problems) relying on a standard heuristic search planner. Schattenberg gives quiet a number of heuristics for hybrid planning problems[10], but they do not yet exploit the hierarchical nature of abstract tasks [L38]. That is, these heuristics do not perform an analysis about the effort that the decomposition of an abstract task will entail; they are thus likely to be less informed than heuristics that investigate the effort imposed by abstract tasks. This lack of information was addressed later by exploiting hierarchical landmarks – which are computed based on analyzing the hierarchy of abstract tasks – as a means for search control [10]. Both Bercher et al. and Bechon et al. exploit the AND/OR structure of the abstract tasks' hierarchy for the construction of heuristics for hybrid planning [7, L19]. These ideas were further extended to obtain an admissible heuristic for both HTN and hybrid planning problems [1].

The Task Decomposition Graph. All of the heuristics and strategies for hierarchical planning proposed in this thesis are based upon the so-called *Task Decomposition Graph (TDG)* [1, 10], a (finite) data structure that represents all of the decomposition methods of the domain, their tasks, and their interplay (for a graphical illustration, consider Fig. 3.1). It is an improvement of the so-called *Task Decomposition Tree (TDT)* by Elkawkagy et al. [L33]. In contrast to the TDT, the TDG can cope with cyclic decompositions without becoming infinitely large. A TDT-like structure was also introduced before by Lotem et al. [L67], which they called *planning tree*. All these structures are closely related to the decomposition tree (see Sec. 2.3) that is tree-like representation of the decomposition methods required to obtain a specific task network/partial plan. In contrast, the TDGs are representing all decomposition methods of the domain that are reachable from the initial task network/partial plan.

A TDG is an AND/OR graph consisting of two types of nodes: task nodes and method nodes. Task nodes are the tasks (primitive or abstract) that are reachable from the initial partial plan. If a task node is abstract, then its methods form its children. Since one is allowed to choose a method during planning, task nodes can be considered OR nodes. Given a method node, the tasks of its partial plan form its children in the TDG. Since all tasks of a method have to be introduced into a partial plan after the respective method is chosen, method nodes are considered AND nodes (cf. Fig. 3.1).

Pruning the TDG. Elkawkagy et al. [L33] show how reachability information about the primitive tasks can be exploited to remove provably irrelevant parts from the TDT. We

[10] In his work, the terminology "heuristic" is not used; he defines plan and modification ranking functions for the hybrid planner PANDA$_1$.

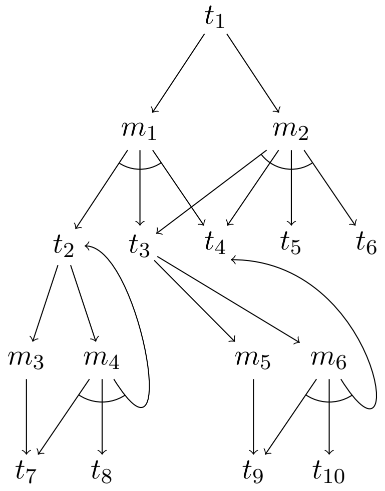


Figure 3.1: Example TDG. The nodes labeled with $t_1, \ldots, t_{10}$ represent tasks. Tasks with outgoing edges (i.e., t_1, t_2, t_3) are abstract. The nodes labeled with $m_1, \ldots, m_6$ represent methods. The AND structure of methods is indicated by a circular arc connecting their outgoing edges.

As described in the next paragraph about landmark-aware strategies, the tasks t_3 and t_4 are local landmarks of t_1, the task t_7 is a local landmark of t_2, and t_9 is a local landmark of t_3. The only "real" landmarks (i.e., tasks that occur on any sequence of decompositions to a solution) are t_3, t_4, and t_9.

(The graphic is a modification of previously published examples [10, Fig. 1] [7, Fig. 1].)

have adopted and improved their technique to be used for TDGs [1]: It first performs a top-down reachability analysis, collecting all primitive tasks reachable from the abstract tasks in the initial partial plan (relying on a parameter-relaxed TDG, where only the names of reachable actions are taken into account, but all reachable groundings). Using only these primitive tasks, a relaxed planning graph is built to identify the set of ground primitive actions that are reachable from the initial state. Then, the TDG is built and restricted to a connected subgraph, in which no method contains an unreachable primitive ground task or an infeasible abstract task. Abstract tasks are infeasible if they either have no method associated (which can happen during pruning the TDG) or if they cannot be decomposed into a primitive set of actions (which can also happen during pruning the TDG due to cyclic method definitions).

The overall pruning technique assumes the typical HTN criterion that does not allow tasks to be inserted arbitrarily. This assumption is manifested by the first step that restricts the planning graph construction to those primitive tasks that are given in the initial partial plan or reachable from its abstract tasks. If the TDG-based heuristics are to be extended to being able to cope with problems that allow task insertion, then this first step is omitted and all primitive tasks are being used for the relaxed reachability analysis. This way, all of the heuristics introduced in the remainder become applicable for the respective problem class in which task insertion is allowed.

Search Strategies and Heuristics Based on the TDG. Traversing a TDG allows to estimate how hard an abstract task is, i.e., which further tasks will be introduced into a partial plan when decomposing it. We will now see various possibilities of how to do so.

Landmark-based Search Strategies. Elkawkagy et al. [L33] exploited the TDT to extract so-called hierarchical landmarks: tasks that occur in any sequence of decompositions leading from the initial partial plan to any solution (e.g., the tasks t_3, t_4, and t_9 in Fig. 3.1). They introduced a procedure that identifies such landmarks and, at the same time, prunes the domain model based on a reachability analysis of the primitive tasks. For the identification of landmarks they perform an over-approximation: they intersect the (ground) tasks of the decomposition methods that belong to the same abstract task. That way, they identify all tasks that will inevitably be introduced when decomposing the respective abstract task. This kind of landmarks was later called *local landmarks* of their parent task to highlight the fact that tasks computed in that way do not necessarily fulfill the requirement that they occur in any sequence of decompositions leading to a solution, but rather in any decomposition rooted in the decomposed task [10]. In the example given in Fig. 3.1, the tasks t_3 and t_4 are local landmarks of t_1, t_7 is a local landmark of t_2, and t_9 is a local landmark of t_3. "Real" landmarks as defined above can be identified by recursively intersecting the local landmarks of all tasks that are part of the initial partial plan [7]. Assuming that t_1 is the only task in that partial plan, we can identify t_3, t_4, and t_9 as such landmarks (that is, the local landmark t_7 is not a landmark).

Elkawkagy et al. [10] exploited the information about which tasks are local landmarks for search guidance. The proposed approach was specifically designed for PANDA$_1$, which uses two different ranking functions: one that orders the partial plans of the search fringe and one that orders the modifications of all flaws (see Sec. 3.1). In our paper, the landmarks are used to rank modifications. Four strategies are proposed, which all base upon so-called *optional task sets*, which are those sets of tasks that are not local landmarks. When considering the example given in Fig. 3.1, the optional task sets of t_1 are $\{t_2\}$ and $\{t_5, t_6\}$, the optional task sets of t_2 are $\emptyset$ and $\{t_8, t_2\}$, and for the task t_3, the optional task sets are $\emptyset$ and $\{t_{10}, t_4\}$. These optional tasks are exploited to prefer one decomposition (modification) before another if the respective number of optional tasks for that modification is smaller than for the other. The general idea behind these strategies is that the number of optional tasks tells more about the refinement effort of an abstract task than the overall number of its methods' tasks, since the non-optional tasks (i.e., the local landmarks) have to be introduced into resulting refinements, anyway. For further details, we refer to the paper [10]. We have later extended these ideas to be used by standard heuristics [7]; instead of ranking different refinement options, we defined a standard heuristic that estimates the goal distance of a given partial plan. This distance does not base upon local landmarks or optional tasks, but rather upon actual landmarks, thereby also giving a lower bound on the refinement effort of a given partial plan [7].

Results. Both the empirical evaluation of Elkawkagy et al.'s landmark-aware strategies [10] (that bas been done with PANDA$_1$) as well as our adaptations to landmark-aware heuristics [7] (that bas been done with PANDA$_2$) was done on four hybrid planning domains. We have compared our techniques with several standard strategies from the literature. There are no significant differences between the different strategies concerning coverage (i.e., number of solved problem instances), so we have shown the required search time (and, in case of our subsequent paper [7], also produced search space) per problem instance. There were no clearly superior strategies/heuristics that outperform all others in all the problem instances, but in the majority of the instances, the proposed strategies and heuristics were among the best-performing ones.

Admissible TDG-based Heuristics. The previously mentioned landmark-based heuristics exploit the TDG in order to compute the number of landmark tasks of a given abstract task as an estimate of the number of further tasks that will unavoidably be added at some point. Relying on just these landmarks will, however, ignore all non-landmark tasks in a TDG. Consider, for instance, a planning problem where no two decomposition methods that belong to the same abstract task have any task in common. As a consequence, no landmarks can be detected and the respective heuristic becomes blind. We have thus developed a generalization of the landmark concept that takes all tasks into account – landmark or not. It exploits the AND/OR structure of a TDG: The refinement effort of a partial plan (an AND node) is given by the sum of the refinement efforts of its tasks, and the effort of an abstract task (an OR node) is given by the effort of its cheapest decomposition method, the effort of which is the effort of its partial plan[11]. We have developed two heuristics that are based upon this idea, one estimates the refinement effort of a partial plan, i.e., the number of required modifications to turn that partial plan into a solution, and the other estimates the cost of a resulting solution plan.

Modification-aware TDG heuristic. The modification-aware TDG heuristic, $\mathrm{TDG}_{\mathrm{m}}$, is a *hybrid planning heuristic* [1]. That is, it is tailored to hybrid planning systems such as PANDA [7] or HiPOP [L19], as it estimates the number of required modifications a hybrid planner needs to perform to turn a search node (i.e., a partial plan) into a solution. It estimates the number of decompositions and causal link insertions. The heuristic is applicable to both hybrid problems without task insertion as well as to HTN problems.

Let $\langle V_T, V_M, E_{T\to M}, E_{M\to T}\rangle$ be a TDG, where V_T and V_M are the task and method vertices, respectively, and $E_{T\to M}$ and $E_{M\to T}$ are the TDG's edges. Then, each node $v_t \in V_T$ and $v_m \in V_M$ in the TDG is associated with an estimate computed as follows:

$$h_T(v_t) := \begin{cases} |pre^+(v_t)| + |pre^-(v_t)| & \text{if } v_t \text{ is primitive} \\ 1 + \min\limits_{(v_t,v_m)\in E_{T\to M}} h_M(v_m) & \text{else} \end{cases}$$

[11] Exploiting the AND/OR structure of abstract tasks in this way has first been proposed independently of each other (and at the same time) by Bercher et al. [7] and Bechon et al. [L19].

For a method vertex $v_m = \langle PS, CL, \prec, \alpha \rangle$, we set:

$$h_M(v_m) := \sum_{(v_m, v_t) \in E_{M \to T}} h_T(v_t) - |CL|$$

According to these equations, we sum over the tasks within the same partial plan, while minimizing over the different options for decomposing an abstract task. The heuristic counts unprotected preconditions and abstract tasks, since for each at least one modification (a causal link insertion or a decomposition) needs to be applied. Please note that we can guarantee finite estimates even if the underlying TDG is cyclic. How this can be achieved is explained in detail in our paper [1]. Please note that the $\mathrm{TDG_m}$ heuristic is actually an improvement of the *Minimal Modification Estimate (MME) heuristic* proposed in an earlier paper [7]. In contrast to $\mathrm{TDG_m}$, the MME heuristic uses a visited list of already processed abstract tasks; while this technique also ensures finite values in the presence of cyclic decompositions, it produces less accurate heuristic estimates compared to the technique used for $\mathrm{TDG_m}$, which uses strongly connected components.

Given a partial plan P that was produced during search, we use the TDG's pre-computed estimates for the heuristic of P:

Let $P = \langle PS, CL, \prec, \alpha \rangle$ be a partial plan. Then,

$$h_{TDG_m}(P) := \sum_{l:t(\bar{\tau}) \in PS} h_T(t(\bar{\tau})) - |CL|$$

Please note that the heuristic proposed in our paper [1] is slightly more general than the one described here. In the paper, we give a lifted variant of the heuristic, where search nodes (i.e., partial plans) may still have variables that are not yet assigned to constants. We give the ground variant here to coincide with the (ground) formalization used throughout this thesis.

Cost-aware TDG heuristic. The former heuristic estimates the number of required modifications to turn a given partial plan into a solution. This sort of goal distance estimate tells nothing about the quality of any solution plan that is found that way, however. An expensive solution might only require a low number of modifications to be found, whereas a cheap solution might require a large number of modifications. To give optimality guarantees on the solutions generated we have adopted the previous modification-aware TDG heuristic to a cost-aware heuristic [1]. Both heuristics follow the very same idea of taking the sum within a method while minimizing over different methods for the same abstract task.

Let $\langle V_T, V_M, E_{T\to M}, E_{M\to T}\rangle$ be a TDG as above. Then, each node $v_t \in V_T$ and $v_m \in V_M$ in the TDG is associated with an estimate computed as follows:

$$h_T(v_t) := \begin{cases} cost(v_t) & \text{if } v_t \text{ is primitive} \\ \min\limits_{(v_t,v_m)\in E_{T\to M}} h_M(v_m) & \text{else} \end{cases}$$

For a method vertex $v_m = \langle PS, CL, \prec, \alpha\rangle$, we set:

$$h_M(v_m) := \sum_{(v_m,v_t)\in E_{M\to T}} h_T(v_t)$$

Let $P = \langle PS, CL, \prec, \alpha\rangle$ be a partial plan. Then,

$$h_{TDG_c}(P) := \sum_{\substack{l:t(\bar{\tau})\in PS \\ t(\bar{\tau}) \text{ abstract}}} h_T(t(\bar{\tau}))$$

Again, we here only give the ground version of the heuristic, whereas the one published can also cope with search nodes that are lifted [1]. Please note that in the TDG$_\mathrm{c}$ heuristic, in contrast to TDG$_\mathrm{m}$, we only retrieve the TDG's estimates for a partial plan's *abstract* tasks. The reason for this is that the costs of a partial plan's primitive tasks are not reflected by the heuristic, but by the cost of that partial plan.

TDG recomputation and incremental heuristic computation. As already noted, the heuristics as described so far are pre-processing heuristics, because the TDG is only built once prior to planning. The heuristics then merely retrieve the TDG's values for the tasks in the current partial plan. However, because the TDG takes the interaction of its tasks to only a limited extent into account, rebuilding it during planning can improve the accuracy of the heuristic [1]. We have given a domain-independent criterion stating in which situations rebuilding a TDG has a chance of creating a different TDG, which differs from its parent TDG due to removed parts that turned out to be infeasible. Recall that we remove all parts from the TDG in which there are unreachable primitive tasks. Since the delete-relaxed reachability analysis is based upon the set of primitive tasks that are reachable from the current search node, we can obtain a different TDG if this set changes – which happens as the result of applying decomposition methods. In all other cases, i.e., if we know (according to our criterion) that TDG recomputation will not result into a changed TDG, we can perform an incremental heuristic computation, where we reuse the heuristic value of its parent node instead of retrieving the required values from the TDG. This is especially important when we enable TDG recomputation, because otherwise we would need to store multiple TDGs (for each of the search nodes) in memory; this way, we only need to store one TDG – similar to the standard variant in which we do not recompute the TDG.

Results. We have performed an empirical evaluation with PANDA, in which we have compared the $\mathrm{TDG_m}$ and $\mathrm{TDG_c}$ heuristics with several heuristics and search strategies from the literature. We run both heuristics with and without recomputation. In addition to comparing coverage, we were particularly interested in the quality of the produced solutions. In our paper [1], we describe the results in detail, so we just briefly summarize the main findings here.

Both search algorithms that use the $\mathrm{TDG_m}$ heuristic, i.e., A* and Greedy-A*, are the best performing system configurations in terms of coverage, each solving 57 of 59 problem instances. Enabling recomputation has no effect on coverage for $\mathrm{TDG_m}$. The performance of the $\mathrm{TDG_c}$ heuristic depends significantly on the chosen algorithm. In combination with the A* algorithm, $\mathrm{TDG_c}$ solves only 52 problem instances. The most successful uninformed system configuration, depth-first (DF) search, solves 53 problem instances; however, the resulting solutions are often of lower quality than than those generated by A* with $\mathrm{TDG_c}$. Whereas the latter guarantees to find optimal solutions, DF search often finds solutions that are up to a factor of 2.09 as large as the optimal one. $\mathrm{TDG_c}$ in combination with Greedy-A* is comparable with the performance of $\mathrm{TDG_m}$. Without TDG recomputation, it solves 56 problem instances and if TDG recomputation is enabled, it solves 57 problems.

Except for (uninformed) uniform cost search, A* in combination with $\mathrm{TDG_c}$ is the only system configuration that guarantees to find optimal solutions. In our empirical evaluation, we can see that all other system configurations (except uniform cost search and breadth first search) also find non-optimal solutions, which are often of significantly worse quality.

The TDG recomputation does never increase the explored search space, but often reduces it – sometimes significantly. Due to the overhead of recomputing the TDG, this search space reduction does not always pay off in terms of runtime, however. In our evaluation, we were able to improve coverage of the $\mathrm{TDG_c}$ heuristic from 56 to 57 due to TDG recomputation. It solves one additional problem instance in the so-called Woodworking domain, in which the search space reductions were quite severe. In one problem instance, the search space was reduced by 83% for $\mathrm{TDG_m}$ and by 93% by $\mathrm{TDG_c}$, which coincides with search time reductions of 46% and 40%, respectively.

Planning is at the heart of any companion-able technical system.

Susanne Biundo, Invited Talk at ICAPS 2016

4 Practical Application

Motivation. Technical systems become more and more complex. The spectrum of a modern technical system's functionality is often diametrically opposed to its level of user-friendliness and ease of operation. Often technical devices have a static user interface and often a poorly written or even incomplete instruction manual. It is common knowledge that most people are not even considering such instruction manuals and those who do, often get frustrated by their quality. We want to exploit the today's scientific capabilities to increase the functionality of technical devices and the way it is presented to its user. More precisely, we want to make them *companion-able*: trustworthy, competent assistants that provide their functionality in an adaptive and truly user-friendly way. *Companion Technology* aims at fulfilling that research vision based on a multitude of different research disciplines – at its heart being AI planning.

With a planning component as a *Companion System*'s main decision making component, we need to face various challenges, all of which can be addressed by relying on hybrid planning:

- *Plans need to be communicated in an adequate manner.* Plans can become quite complex. So, presenting such a complex plan at once might confuse the user (or even be impossible due to its size). Instead, one might present it via showing one plan step at a time. However, even then we need to decide which order suits the respective user the most, and how a chosen plan step is presented (i.e., on which level of abstraction it is presented and which modality should be used).
- *Plan execution should be monitored for the identification of errors and failures. If required, plans need to be repaired.* One of the biggest challenges when applying technology in practice is the influence of "the real world". Many assumptions taken for a model do not hold under real-world conditions. To have a robust system, we need to be able to cope with execution errors, particularly with *unforeseen* failures.
- *Plans and planning decisions should be explainable.* This allows a user to ask questions about the system behavior or any instruction presented to the user. This capability directly contributes to the system's transparency and the user's trust in that system. Plan explanations become particularly important if plans change due to the result of repairing them.
- *It should be possible to cooperatively develop plans together with the user.* Developing plans cooperatively allows a user to *directly* influence the plans generated

by the system and therefore ensures fulfilling the user's constraints and preferences. (While our planning procedure does support extensions for this form of mixed-initiative plan generation, it is not within the focus of this thesis.)

Summary of Core Contributions. We have given the very first overview article for *Companion Technology* [3]. Apart from explaining the vision of *Companion Technology*, we mention further related surveys, explain what and how related research disciplines contribute to the technology, and give a comprehensive list of application areas in which the respective systems may enrich their functionality and user friendliness by *Companion Technology*. Due to the high interdisciplinarity of *Companion Technology*, there are countless research projects that address different aspects of the overall research vision; we have given a comprehensive overview of these research projects and explain similarities and differences to *Companion Technology*.

To enrich a technical system's functionality by the use of AI planning, we have identified various so-called user-centered planning capabilities [2, 11]. These are the (fast) generation of plans, presenting them in an adequate way, carrying them out (which includes monitoring their execution), repair them if execution failures occur, and explain them to the user if questions arise. In our work, we discuss the importance of these capabilities for *Companion Systems*, give a short but sound overview of how they work, and show how these capabilities serve as the basis for providing advanced user support in a broad variety of application scenarios.

We have developed a prototypical *Companion System* [2, 5, 6, A5, A22] that implements the user-centered planning capabilities mentioned above [2, 11]. The communication of its various components (which implement techniques stemming from knowledge representation and reasoning, AI planning, dialog management, and interaction management) bases upon a generic system architecture [6]. To illustrate how implementing this architecture for a specific application scenario results in a flexible and situation-adaptive assistant, we developed an assistant for setting up a complex home entertainment system. The system bases upon a declarative (planning) model of the available cables and devices in order to fully automatically find a course of plug actions that the user simply needs to execute in order to set up his or her system. The system implements all of the before-mentioned capabilities of plan generation, finding a user-friendly linearization of its plan steps, presenting these steps in an adequate way, monitoring their execution to find repaired plans if unforeseen circumstances prevent standard plan execution, and explaining presented plan steps (i.e., plugin instructions) if the user wants to be informed why he or she should execute them. In an empirical evaluation with 59 test subjects, we evaluated how our assistant is perceived in general, and how plan explanations influence this perception in particular [6].

The above-mentioned core contributions are published in the following publications:

[2] **P. Bercher**, D. Höller, G. Behnke, and S. Biundo. "User-Centered Planning". In: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by S. Biundo and A. Wendemuth. Cognitive Technologies. In print. Springer, 2017. Chap. 5, pp. 79–100. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4_5

[3] S. Biundo, D. Höller, B. Schattenberg, and **P. Bercher**. "Companion-Technology: An Overview". In: *Künstliche Intelligenz* 30.1 (2016). Special Issue on Companion Technologies, pp. 11–20. DOI: 10.1007/s13218-015-0419-3

[5] **P. Bercher**, F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nothdurft, F. Honold, W. Minker, M. Weber, and S. Biundo. "A Planning-based Assistance System for Setting Up a Home Theater". In: *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, 2015, pp. 4264–4265

[6] **P. Bercher**, S. Biundo, T. Geier, T. Hörnle, F. Nothdurft, F. Richter, and B. Schattenberg. "Plan, Repair, Execute, Explain - How Planning Helps to Assemble your Home Theater". In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*. AAAI Press, 2014, pp. 386–394

[11] S. Biundo, **P. Bercher**, T. Geier, F. Müller, and B. Schattenberg. "Advanced user assistance based on AI planning". In: *Cognitive Systems Research* 12.3-4 (Apr. 2011). Special Issue on Complex Cognition, pp. 219–236. DOI: 10.1016/j.cogsys.2010.12.005

4.1 Companion Technology

Most of the technical devices that we use in our daily lifes are still quite simple concerning their underlying control logic while still being too complex to be operated by a standard user that just wants to use the device without being particularly interested in how it works internally. As a consequence, most users are not aware of the full spectrum of a modern technical device's functionality. To address this lack of intelligence in today's systems, research is done aiming at creating *cognitive* technical systems. Cognitive technical systems are technical systems that show cognitive capabilities such as planning and reasoning. Most importantly, they have a knowledge base that allows planning and reasoning, so that they "know, what they are doing" [L46]. Several recent research projects are concerned with research in that general area. The most recent ones are *CoTeSys (Cognition for Technical Systems)* [L32, L46], *CITEC (Cognitive Interaction*

Technology) [L34], and *A Companion Technology for Cognitive Technical Systems* [L9].

Companion Technology extends the idea of cognitive technical systems by addressing various additional capabilities that differentiate *Companion Systems* from other cognitive technical systems: *Companion Systems* are cognitive technical systems that adapt their behavior completely to the individual user. For this, *Companion Technology* enhances the functionality of a technical device to provide automated assistance in using it: Instead of the user having to adapt to the system, the system adapts to its user. To achieve this goal, research focuses on individuality, adaptability, availability, cooperativeness, and trustworthiness [L9].

In the literature, the term "companion" is used in many different flavors, from being meant in the literal sense (like a toy robot with only a very limited amount of interaction capabilities) to the definition provided above. For a comprehensive overview of the current state of the art and its history – from cognitive technical systems to *Companion Systems*, we refer to our survey on *Companion Technology* [3]. It includes related research projects, involved research disciplines, and possible application areas of the technology. The wide spectrum of possible applications and research directions can also be seen in a recently published KI special issue[1] on *Companion Technology*, for which we were the guest editors [A9]. A comprehensive overview of the latest research results that emerged from the project *A Companion Technology for Cognitive Technical Systems* is given in a recently published book [L3]. In this book, we also explain the user-centered planning capabilities that were developed within the context of this project and show how they can be exploited as the basis for intelligent assistants [2, A3, A5].

4.2 User Assistance Based on AI Planning

Here, we show how AI planning can be used as a basis for providing automated support to a user for solving tasks that can be formulated as planning problems. Due to their general nature, planning problems occur often in practice – whenever a sequence of actions needs to be carried out in order to achieve a certain goal. Some examples are robot control, project or (space) mission planning, or operating technical devices (as we will see in our prototypical *Companion System* described below). When planning technology is applied in a setting that involves human users, it is not only important that (good or preferred) plans are generated fast, but various additional requirements arise. We deem hybrid planning as adequate in this context [A19], since it allows to address these requirements based on so-called user-centered planning capabilities [2, 11], which we summarize below.

[1] The special issue is available at `http://link.springer.com/journal/13218/30/1/page/1`

Planning-based Assistance Systems (Related Work). Despite the great potential and large number of possible application scenarios for planning, the technology has not yet become a standard technology to be applied in practice or even industry. To give some examples where AI planning has already been successfully applied practically, we here give a short overview starting with a system from 2002. Additionally, we want mention (the website of) the Special Interest Group for Applications of AI Planning and Scheduling (SIGAPS), which "aims to widen awareness of AI Planning and Scheduling technology, promote its application outside academia, and provide resources for researchers interested in tackling application problems" [L1].

The assistance system by Pollack bases on POCL planning techniques [L60]. In this regard, the system's planning approach is highly related to ours, since the hybrid planning formalism extends POCL planning by means of a task hierarchy (see Sec. 2.4). Pollack's system is designed to assist elderly people and people with cognitive impairments in routines of their daily life. Such routines include very fundamental ones, such as eating and drinking or using the bathroom, but also less fundamental ones, such as managing medicine or housekeeping. The general goal is that people relying on the developed assistant can remain their independence for a longer time and therefore stay in their own homes longer. The assistant runs on an autonomous mobile robot. One of the central functionalities of the system is to remind its user about forgotten routines. For this, it features a model of time: tasks can have a duration and a planned point in time when they are supposed to start.

Closely related to this endeavor is the work by Boger et al. [L49, L53], who develop techniques to assist people with dementia in daily routine tasks. While the former approach focuses on maintaining and complying to the user's schedule, this approach focuses on assisting to carry out the respective tasks themselves in a correct way, i.e., that no sub actions were forgotten or carried out in a wrong order. The task that is used as an example is washing hands, which consists of several sub steps, such as turning on and off the water and taking the soap. As underlying planning model, they use Markov Decision Processes (MDPs) and the generalization thereof to Partially Observable MDPs (POMDPs).

The work by Beetz et al. is also motivated by assisting elderly people by duties of daily life [L27]. By relying on their techniques, autonomous service robots should achieve "home chore task intelligence". These robots should be able to perform a series of different tasks and are controlled by so-called "cognition-enabled robot control programs", which use cognitive mechanisms, such as learning, reasoning, and planning. They use a hierarchical planning approach that they developed for their specific requirements.

Another real-world planning-based system is described by Petrick and Foster [L24]. Their system is not directly aiming at providing assistance in every-day tasks, however. They apply planning to control a robot with basic social skills that is able to interact with multiple humans in a simple bartending domain. They show how they infer social states from low-level sensors by relying on vision and speech input. They use a non-hierarchical planning approach that can cope with partial observability and that supports so-called sensing actions which allow to acquire missing information (e.g., the task of asking whether a customer wants a drink).

The system by Bernardini and Porayska-Pomsta is designed to help children with Autism Spectrum Conditions to acquire social communication skills [L22]. For doing so, it maintains a user model of the child that reflects its current cognitive and affective state. Their system is a virtual agent the child can interact with. Its behavior is based upon automatically generated plans, created by a POCL planner. As our system does, their system also receives direct user input via touch gesture and sensory information. They also have an execution monitor that checks whether actions were successful or not; if not, plans can be repaired.

User-Centered Planning Capabilities. As motivated in the beginning of this section, the capability to generate plans is not the only one required for the design of flexible *Companion Systems*. We need to address various further issues, such as:

- Which plans should be found? Is optimality important for the interaction with human users? Are there preferences that need to be respected? Are there optional goals?
- How to present the generated plans? How to cope with the partial order of plans? Can we exploit the information that is given within those plans, i.e., the causal links and the underlying task hierarchy?
- Can we cope with execution errors?
- How to ensure the user's trust in the solution being carried out? Can we answer questions about the purpose of presented instructions?

We have developed planning techniques that address these questions arising in the context of planning with or for human users. Hence, we refer to them as user-centered planning capabilities. Prior to the development and implementation of most of these capabilities and, consequently, before their deployment in an actual running system, we described our vision of how they can be exploited as the basis to provide assistance in a variety of tasks – exemplified at the example of operating a modern smart phone [11]. About six years later, after all of those capabilities have been fully developed and published in individual papers, and after their deployment in a running system (see below), we "re-visited" the topic of *user-centered planning* thereby explaining how these techniques work and how their integration and interplay allows advanced assistance in a broad variety of possible application scenarios [2]. Here, we give a brief summary of these techniques. For doing so, we will present them in the same order they become important in a system that exploits them, starting with *the generation of plans*, *presenting them to* *the user*, *repairing them if execution failures are detected*, and *answering questions upon user request.*

Generating Plans. In Sec. 3.1, we have introduced our planning procedure. Most importantly for the sake of providing user assistance, we want to highlight that it is a hierarchical planner that generates partially ordered plans, in which the causal structure is explicitly represented. Plans being only partially ordered (while guaranteeing that any of its linearizations is executable and satisfies the user's goals) brings us more flexibility with respect to execution: we can thus flexibly decide which of its linearizations is most appropriate for the individual user in the current situation. This will be described in the next part that is about linearizing plans. Both the causal structure and the hierarchy can be exploited for such linearization strategies as well as for generating plan explanations. Note that the causal structure of a plan can also be inferred from a totally ordered action sequence [L31]; and also the partial order can be inferred [L30]. Thus, in principle, one could also rely on a planning procedure that does not already generate plans showing these desired properties.

The planning system described in Sec. 3.1 can further be provided with admissible heuristics [1, 8] to find optimal plans thereby avoiding redundancies in plans that might confuse the user (such as performing a plug action, just to unplug it again later). Concerning finding preferred plans, i.e., plans that satisfy additional user requirements such as optional goals that are not mandatory, but increase the user's satisfaction with the plan, we have proposed such a heuristic that works both for POCL and hybrid planning problems [A24]. Other approaches for finding preferred plans in hierarchical planning also rely on specialized heuristics for preferences [L39, L43].

Linearizing Plans. After a solution plan has been found, it has to be communicated to the user in an adequate way. Presenting the entire plan at once does not seem an appropriate solution, because it might be too large to comprehend. At least, techniques would have to be developed that present an abstraction thereof. For this, the task hierarchy can be exploited to find adequate abstractions of different levels. As a first step, we chose to present all steps of a plan one by one.

Since plans are only partially ordered, the question arises which of the possible linearizations are most adequate for the respective user in the current situation [2, A21]. From our solution criteria, we get that *all* linearizations are correct in the sense that they are executable and achieve the user's goals. However, there might still be a huge number of possible linearizations of the given plan and some of them might be more useful or intuitive than others. So, the question arises about the essence of so-called user-friendly linearizations. As an example, consider the task of connecting various devices with cables. Assume that there is a cable in "T"-shape having two plugs at one end (like CINCH audio and CINCH video) and one plug at the other (like SCART featuring audio and video). The model foresees one individual plug action for each of its plugs, resulting in three independent actions. Clearly, any order can be performed and other actions can be executed in between. However, it would be odd to plug in just one of the two plugs that belong together, perform some other – completely different – actions in between and then come back for the other plug action(s) of the cable. To the best of our knowledge, we are the very first addressing this issue of finding user-friendly linearizations [2, A21]. We discussed what linearizations can be considered more user-friendly than others and have come up with three different domain-independent techniques to find them. One relies on the similarity of the constants in the parameter-lists (idea: two actions are manipulating the same object), one on two actions sharing a causal link (idea: one action is causally relevant for the other), and one takes the distance in the decomposition tree/task hierarchy into account (idea: actions are related if they are close to each other in the task hierarchy).

We are currently collaborating with psychologists to evaluate, among others, how humans are solving problems to find the most adequate way to assist them [A1], e.g., by adjusting our user-friendly linearization strategies to coincide with the findings from our empirical studies.

Communicating Plans. After we have found a most-adequate linearization, we present it step by step [2, 6, A5]. For this, each action has a dialog model attached. That model makes use of all information and further media that is specified along with that action, such as pictures and videos (which can be associated to the action parameters). It thus decides, taking user knowledge into account, how the current plan step is displayed [A5]. Later, in Fig. 4.3, we will see an example taken from our assistant for setting up a home theater.

Repairing Plans. When plans are carried out in the real world, it is quite likely that something will happen that prevents that plan from being executed in the way it was intended. In our running example of setting up a home entertainment system, a cable might be broken, which the user only recognizes during plan execution. We are thus in need of a repair mechanism that can cope with unforeseen events that influence plan execution [2, 11]. Such a plan repair procedure for hybrid planning was first introduced by Bidot et al. [L41] and improved later on [6, 11]. The repair mechanism is able to cope with unforeseen state changes, for example, when a cable should be available according to the sequence of actions applied so far, but during execution it gets revealed that it is not. The causal links in a plan can be exploited for deciding whether repair must be initiated or whether we can proceed executing the plan despite the unforeseen changes to the world state. If the plan needs to get repaired, a new plan is found in which this altered current state is taken into account. Because of the solution criterion that demands that any solution plan needs to be a refinement of the initial partial plan, we need to make sure that all actions that were already executed prior to the unforeseen state change are again part of any repaired solution. This makes the difference between re-planning (which starts planning from scratch after each execution failure) and plan repair (which takes the already found solution into account for addressing execution errors) more subtle in the hierarchical planning setting compared to non-hierarchical planning, where this distinction is more clear. We discuss this matter in detail in our latest publication about user-centered planning [2]. Finding repaired solutions relies on so-called plan obligations, which are additional constraints posed on solutions. These obligations are implemented as new flaw classes in our hybrid planner PANDA [6].

Explaining Plans. Explaining system behavior is a key capability of intelligent systems that provide support to their users, since unexpected or non-understandable behavior can impact a user's trust in a negative way [L12, L83]. Plans can not only serve as basis for instructions presented to the user, but also for directly controlling system behavior (like switching a light on or off) [2]. Since both types of plans/actions have the same underlying structure, any plan explanation technique will be applicable to both types.

In our first work on plan explanations, we outlined what a user might possibly be interested in to know about a given plan and how we can make use of causal links and the hierarchical structure to answer such questions automatically [11]. Seegebarth et al. [L28] have then taken these ideas forward and showed how such questions can be answered automatically. Possible questions include the necessity of an action (e.g., being phrased as "Why should I do this?") or the necessity of a certain order (e.g., being phrased as "Why should I do this now? Can't I do X first?"). The technique to answer such questions is based upon a proof in an axiomatic system that encodes the problem to solve (i.e., the initial partial plan), the given solution, the history of its plan modifications, and general justification rules stating why some action might be necessary in a given partial plan. The latter exploits both the task hierarchy (a task is regarded necessary if it is introduced as the consequence of decomposing an abstract task) and the causal structure (a task is necessary if it provides a causal link for another action). The proof can be regarded a *formal* explanation to the raised question. Since it corresponds to a sequence of axioms, it can be directly translated into natural language [2, 6, L28].

To give an example, we make use of the assembly assistant for setting up a home theater. Consider that the user asks about the purpose of a currently presented plug action. By analyzing the causal structure of the plan, the respective formal proof can be translated as follows. *"This step serves the goal to transmit the video signal of the blu-ray player to the TV. To this end, the video signal of the blu-ray player is transmitted over the HDMI-to-DVI adapter and the HDMI-to-DVI cable to the amplifier. From there it is transmitted over the video-cinch cable to the TV."*

We have done an empirical evaluation with 59 test subjects to study the usefulness of plan explanations (and the explanation mentioned above is one of the explanations presented to the test subjects) [6]. Since this study was done in the context of evaluating our assistant for setting up a home entertainment system, we summarize our findings below – after we have introduced the respective system.

Up to this point, we have seen all the main capabilities that allow to construct an assistant system that provides support to a user in a large set of possible application scenarios. We will now see one such system that has been implemented practically. The system's capabilities encompass all the ones mentioned above and complements them by further capabilities stemming from knowledge representation and reasoning, and dialog and interaction management.

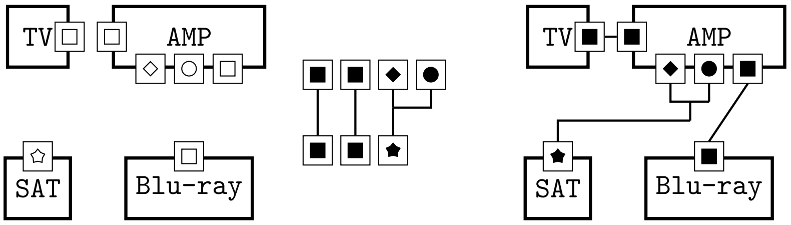
A Planning-based Assistant for Setting Up a Home Entertainment System. We have developed and implemented a generic system architecture to provide automated planning-based assistance in wide variety of possible application scenarios [6, L23]. The paper by Bercher et al. [6] describes the planning-related aspects of that architecture in detail, whereas the paper by Honold et al. [L23] lies emphasis on the user interaction part. That architecture is completely domain-independent – that is, its modules only rely on the different models and further data (such as videos or pictures), but it is not specific to a certain application area or device. Any system relying on that architecture provides the following key capabilities for generating a course of instructions that, if followed by the user in that order, fulfills his or her goals. There are planning modules for all of the user-centered planning capabilities that we have mentioned above. That is, in case something unforeseen happens during the execution of these instructions, the system can adequately react to these failures by providing an adapted course of instructions[2]. The system can explain its own behavior, i.e., for any presented instruction the user might ask about the purpose of it, which then gets explained via natural language (both in written as well as spoken text). In addition to the various modules for the user-centered planning capabilities, there is a central knowledge base that stores and processes all central information that is relevant for more than one system module. Further modules from dialog and interaction management are responsible for presenting the instruction to the user in an adequate manner (i.e., they choose how to present it and which modalities and hardware to use).

To illustrate how our approach can be applied in practice, we have implemented a prototypical *Companion System* for providing support in the task of setting up a complex home entertainment system [5, 6, A5, A22]. A short overview of the complete system (including a comprehensive list of all publications that were done in the context of developing this system) is given in our most-recent description about the system [A5].

The Assembly Task. For our example scenario, we modeled a specific set of available devices, which are known to the system in advance[3]. In our setting, the user's home

[2] Adequate reaction to execution failures can only be guaranteed if the system is aware of these failures. In our prototype, all unforeseen changes have to be reported to the system by the user.

[3] In a more general system, the user would be able to select the devices or cables that he or she possesses. This can trivially be implemented by providing the system with the required model(s) and let the user choose the available hardware. This has not yet been integrated into our demonstration system, however.



(a) The task: an assortment of unconnected devices and cables.

(b) A solution: the devices are properly connected.

Figure 4.1: The figure schematically illustrates the available devices, some of the available cables, and how they are compatible with respect to each other: The TV, amplifier, satellite receiver, and Blu-ray player each have various female ports, where □, ☆, ◇, and ○ denote HDMI, SCART, cinch video, and cinch audio ports, respectively. Black ports on cables ■ denote male ports. There are two HDMI cables and one SCART-to-cinch-AV cable. The devices should be connected in such a way that the video signals of the Blu-ray player and the satellite receiver reach the TV. The respective audio signals should be transported to the amplifier receiver, which is connected to speakers. (The graphic is a slight modification of the one given by Richter and Biundo [L5, Fig. 1]; see also the thesis of Richter [L4, Fig. 1.1].)

theater consists of four devices: a television, an audio/video receiver (also referred to as amplifier), a blu-ray player, and a satellite receiver. For the system to be fully functional, the video signals of the blu-ray player and the satellite receiver need to be transmitted to the television and their audio signals must be transmitted to the audio/video receiver, since the audio speakers are attached to it. For doing so, the user has various cables, adapters, and adapter cables available. We give a graphical illustration in Fig. 4.1.

While that assembly problem seems easy at first glance, it is actually quite demanding for many users. First, it highly depends on expert knowledge in this particular domain. Without such knowledge about the available hardware, the high complexity of some of the devices might act as a deterrent to some users. To get an impression of the hardware's complexity, we depict the back of the modeled audio/video receiver in Fig. 4.2. In Fig. 4.3, we see how the user gets supported in using this device.

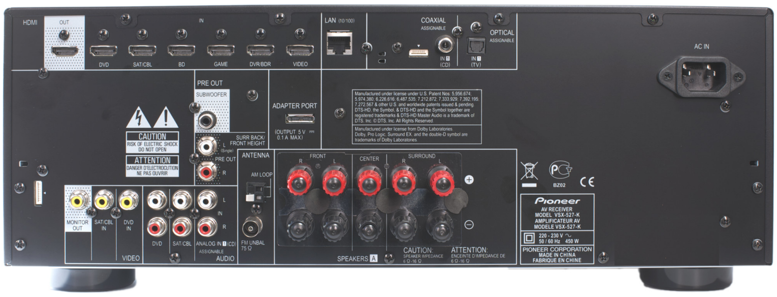
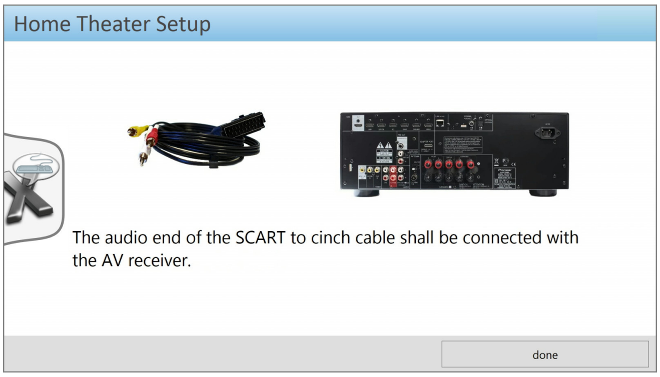


Figure 4.2: The figure shows the back panel of the amplifier used for our assistant. (The graphic shows Fig. 2 published previously [6], but with removed bar code.)



Home Theater Setup

The audio end of the SCART to cinch cable shall be connected with the AV receiver.

Figure 4.3: The figure shows how a plug action is presented as a detailed instruction to the user. It shows the involved cable, the involved device, and it highlights the device's ports that need to be used for plugging the cable into the device. The "X" on the left side opens a menu that can be used for asking about the necessity of the current instruction or for reporting execution errors. (The graphic shows Fig. 1 published previously [5], but with removed bar code.)

To successfully complete the task, one must know which cable transmits which kind of signal (and, given quality is an issue, whether it is analog or digital). For example, some cables are either audio or video cables, others can be used either as audio or video, and others allow both signals to be transmitted at the same time. The task also shows some combinatorial aspects, which become apparent if the number of available cables gets limited – in such a case, it might be necessary to use adapters to a large extent. The task is described in more detail in our paper; there, we also give an excerpt of how it is modeled [6]. A non-deterministic version of this assembly task has also been modeled, relying on a Partially Observable HTN (POHTN) planning approach, which extends Partially Observable Markov Decision Processes (POMDPs) by a task hierarchy, developed by Richter [L4, L5]. This model is not integrated into our system, however.

The System. We have set up and tested our system in an office environment. The assistant runs on a separate device to instruct the user how to set up his home theater. In our prototype system, there are two devices, the user can interact with. Both feature speech and touch input; one of them further features input via pointing gestures. The system is equipped with laser-range-finders for user localization; depending on the user's position, the system can decide which of the devices to use for interaction. The respective setting is shown in Fig. 4.4.

The user gets supported by a sequence of detailed instructions. Fig. 4.3 shows how such an instruction looks like. It shows which of the SCART-to-CHINCH cable's plugs to put into which of the amplifier's ports. To get a more detailed impression of how our assistant works, we refer to a video that we have produced [A22]. It introduces the system and the underlying scientific techniques. The target audience of the video is the general public, so the scientific techniques are only explained on an abstract level (in English).

Evaluation. In our paper about the planning aspects of our assistant [6], we have performed an empirical study with 59 test subjects. The study was not primarily intended

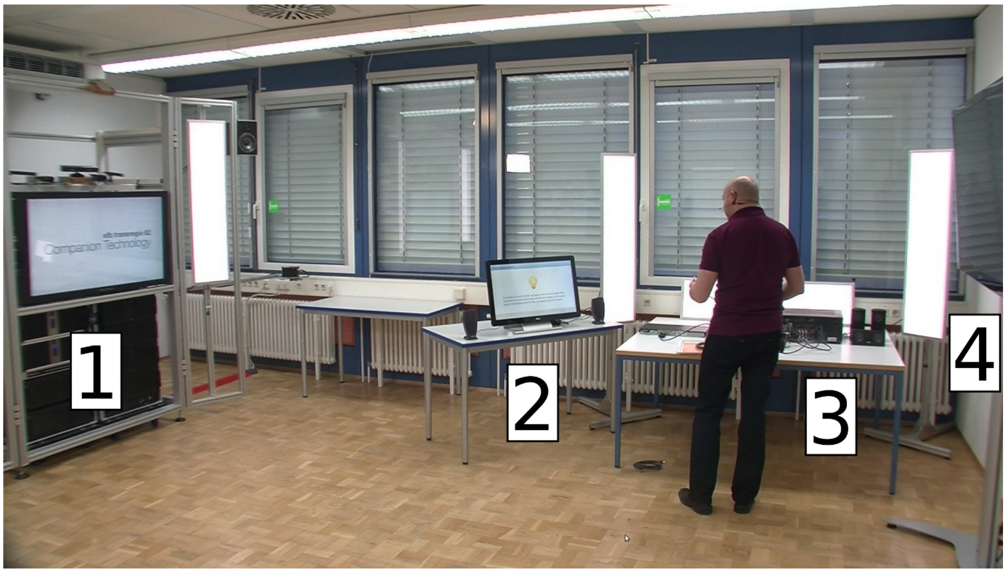


Figure 4.4: The figure shows the setting in which we have tested our assistant. The home entertainment system can be seen on the right side: (3) shows the user interacting with the blu-ray player, satellite receiver, and amplifier. On its right side, (4) shows the television. The assistant runs ons separate devices. (1) shows the main interaction device, which is a large touch screen that also features input via pointing gestures (for which it is equipped with a set of cameras) and speech input. (2) is an additional device that also features input via touch and speech. On the right side of the main interaction device, we can see one of two laser-range-finders for localization of the user. (The picture is an edited version of a screenshot taken from a video describing our system [A22]; see `http://companion.informatik.uni-ulm.de/ie2014/companion-system.mp4` at approx. 18:46/20:26.)

as an evaluation of the system as a whole, but to test the influence of plan explanations on the users' confidence in the correctness of the presented solutions. For this, we did a controlled, randomized trial, where we have split all subjects, without them knowing, into two groups. One was simply instructed to follow the presented instructions, whereas the other group was presented a plan explanation for two of the presented instructions. For reproducibility of our study, we did not use our actual system, but a seemingly interactive HTML5 slide show that corresponded to a fixed course of actions from our system. The results were giving a mixed picture. On the one hand, we were not able to confirm our hypothesis that the presented plan explanations foster the users' confidence in the correctness of the presented solutions. In fact, the control group (which was not presented any explanations) has a slightly lower confidence than the treatment group. This difference was not statistically significant, however. We do believe that this result can mainly be attributed to the experimental condition: the treatment group was presented the explanations without asking for this (and without any necessity from the point of view of the test subjects), so their presentation might have confused the respective test subjects. On the other hand, the overall perception of the system was perceived very positively – by both groups. Despite the slightly negative results for the group that was presented the plan explanations, we also received some positive comments by the respective test subjects, such as *"explanations were good and useful, but the presented version was confusing due to the bad, automated voice"* and *"the explanations seem to be unnecessary at first glance, but they increase the understanding of what one does and strengthen the credibility of the system"*. We also received several positive remarks about the system, such as *"assists in a useful way"* [6]. We also found some interesting results concerning the question for whom such a system might be more appealing. We found that women rated the overall system better than men; also, the degree to which people consider themselves unskilled in that application domain (we asked them about their confidence that they would succeed in the task while only using instruction manuals) predicts how much they like the system in a linear regression.

Conclusions. We have developed a domain-independent architecture that serves as the basis for the construction of (planning-based) assistance systems. It integrates all of the before-mentioned user-centered planning capabilities with advanced dialog and interaction management components to enable multimodal communication of the system. In particular, such a system is able to:

- automatically generate plans that achieve a user's goals,
- present these plans in a step-by-step fashion exploiting the capabilities of advanced dialog and interaction management for multimodal interaction,
- repair plans in case of execution failures, and
- to give justifications about the purpose of any instruction presented to the user.

Possible Extensions (Ongoing and Future Work). There are mainly two directions of research that are directly motivated by our application system and our endeavor of providing user support. The first one is concerned with modeling issues in systems as ours that rely on different kinds of models stemming from different research disciplines. The second is about directly integrating the user in the plan generation process.

In complex knowledge-based systems like the one proposed above, it is of crucial importance that there are no misconceptions, redundancies, or inconsistencies with/between the different models. If there were, maintaining that knowledge becomes quickly intractable and, in the worst case, the system might not work correctly anymore – or not at all. Ensuring these consistencies and eliminating redundancies was a quite tedious task in the development of our system and there was an inherent need for addressing that issue. We have addressed it by an integrated approach that allows to use just a single unified model – knowledge models for the system's individual components are automatically generated from that centralized model [A2, A15, A16]. The generation of the individual models for the planning and dialog components is done using automated reasoning relying on a centralized knowledge model in the form of an OWL ontology [L40]. A key idea to make this work was to exploit the correspondence between subsumption in ontologies and decomposition in hierarchical planning. For this, also tasks (both primitive and abstract) are represented within the ontology. Relying on inference mechanisms, we can further exploit the before-mentioned correspondence to infer new decomposition methods for the planning model. For any such inferred method, the ontology reasoner provides us with further arguments *why* some task is a refinement of an abstract task (which are basically the subsumption axioms that allow to infer the method); standard plan explanations [2, 6, 11, L28], on the other side, can only state *that* one task is within a decomposition method of another – without being able to give further justifications why this is the case. We are currently working on extensions to this approach that focus on the automatic generation of those parts of the planning model that describe world objects and their properties [A6]. To this end, all objects that the planning model's tasks potentially manipulate are described in an ontology using standard taxonomic structures. Properties and interconnections between these objects are represented there as well. This information is automatically incorporated into the planning model.

The second direction of ongoing research that is motivated by the construction of *Companion Systems* is the direct integration of the user into the planning process – this is called *mixed-initiative planning (MIP)*. MIP is often a more adequate choice compared to solving planning problems without direct user involvement, because it allows the user to directly alter plans under development in the way he or she desires. An alternative to MIP is to specify the users' preferences, according to which suitable plans are generated. However, sometimes the users' preferences are not fully known to them or simply not easily expressible in the underlying planning language. For these reasons, MIP techniques were successfully applied in several NASA missions [A11]. In a series of publications, we have demonstrated how a dialog manager can be integrated into a MIP system to interact with a human user incorporating his or her wishes into the currently presented partial plan under development [A2, A3, A4, A18]. These systems are implemented as extensions to our Panda planning system. More specifically, we let the user choose between different refinements (decomposition methods) of a given partial plan (abstract task). In a more general setting, a user would also be allowed to make *changes* to a current partial plan, such as altering the partial order or including, removing, or interchanging tasks. We have investigated the computational complexity of such change request in the hierarchical planning setting [A8], but did not yet implement them in a running system.

5 Conclusion

This thesis is concerned with *hybrid planning* – from theory to practice. Hybrid planning fuses hierarchical task network (HTN) planning with concepts known from partial order causal link (POCL) planning. The approach is motivated by the endeavor of creating planning-based *Companion Systems*, cognitive technical systems that flexibly assist their human users performing complex tasks [3, L9]. In this context, the thesis makes contributions along three lines of research, *Theoretical Foundations* (Sec. 2), *Search and Heuristics* (Sec. 3), and *Practical Application* (Sec. 4). We summarize the main contributions below; for detailed summaries, we refer to the beginning of the respective sections.

Theoretical Foundations. One of the thesis' most foundational contributions is a novel formalization of HTN planning problems [12]. The formalization has proved successful for investigating several computational properties of hierarchical planning, most notably regarding the plan existence problem. It was hence used as a basis for such investigations both by us and other authors as well [4, 12, A7, A8, A10, A12, A13, A14, A20, L2, L7, L8, L14, L18, L26]. Fragments of our formalization were also used for purposes other than proving foundational properties, such as learning task hierarchies [L11, L16] or for providing an overview for HTN planning [L15]. A second key contribution is studying the impact of *task insertion* on the computational complexity of the plan existence problem [12]. We have shown that the capability of task insertion limits the impact of recursive method definitions, making the respective problem class, called HTN planning with task insertion (TIHTN planning), decidable. Investigations of task insertion were taken further later on [A10, A13, A14, A20, L7, L8, L18]. As a third key contribution, we extended our HTN planning formalization to a novel formalization of hybrid planning [4]. Relying on this formalization, we were able to prove the first complexity results for hybrid planning. We showed that most of the hardness results for the plan existence problem in HTN planning also hold in hybrid planning. We further transferred an HTN planning result concerning the plan verification problem to hybrid planning, thereby showing that it is $\mathbb{NP}$-complete to verify whether a given partial plan is a solution to a given planning problem.

Search and Heuristics. To exploit AI planning in real-world applications, well-informed heuristics are required to solve the given problems as quickly as possible. Whereas there are several well-informed heuristics for classical, non-hierarchical planning, there are only quite a few in hierarchical and POCL planning. Based on a theoretical analysis, we created the prerequisites to exploit heuristics from classical planning in the POCL setting [8] and identified what kind of problem relaxation needs to be performed in order to obtain a tractable problem class and respective heuristic [9]. For the latter, we showed that the plan existence problem for (partially) delete-relaxed POCL planning problems is $\mathbb{NP}$-complete. We have further developed heuristics and search strategies for HTN and hybrid planning [1, 7, 10], including the first admissible heuristic for these problem classes which thereby allows to find optimal solutions heuristically [1].

Practical Application. We have presented a prototypical *Companion System* [5, 6, A5, A22] to illustrate how hybrid planning can be deployed as the basis for systems that provide automated user assistance. The system assists its users in the task of setting up a home theater. For such a system to act as a flexible, robust assistant, we identified and realized various user-centered planning capabilities [2, 11], all of which are implemented within our assistant. They encompass the generation of plans, the user-friendly linearization and presentation of plans, their repair in case of execution failures, and the explanation of plans.

6 References

The references are divided in three different lists: the core contributions of the thesis, related work by the author, and related work from the literature.

6.1 Core Contributions

The references listed below are the core contributions of the thesis. They are also included as full PDF in Sec. 7.

[1] **P. Bercher**, G. Behnke, D. Höller, and S. Biundo. “An Admissible HTN Planning Heuristic”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*. AAAI Press, 2017, pp. 480–488. DOI: 10.24963/ijcai.2017/68.

[2] **P. Bercher**, D. Höller, G. Behnke, and S. Biundo. “User-Centered Planning”. In: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by S. Biundo and A. Wendemuth. Cognitive Technologies. In print. Springer, 2017. Chap. 5, pp. 79–100. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4_5.

[3] S. Biundo, D. Höller, B. Schattenberg, and **P. Bercher**. “Companion-Technology: An Overview”. In: *Künstliche Intelligenz* 30.1 (2016). Special Issue on Companion Technologies, pp. 11–20. DOI: 10.1007/s13218-015-0419-3.

[4] **P. Bercher**, D. Höller, G. Behnke, and S. Biundo. “More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks”. In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. IOS Press, 2016, pp. 225–233. DOI: 10.3233/978-1-61499-672-9-225.

[5] **P. Bercher**, F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nothdurft, F. Honold, W. Minker, M. Weber, and S. Biundo. “A Planning-based Assistance System for Setting Up a Home Theater”. In: *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, 2015, pp. 4264–4265.

[6] **P. Bercher**, S. Biundo, T. Geier, T. Hörnle, F. Nothdurft, F. Richter, and B. Schattenberg. “Plan, Repair, Execute, Explain - How Planning Helps to Assemble your Home Theater”. In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*. AAAI Press, 2014, pp. 386–394.

[7] **P. Bercher**, S. Keen, and S. Biundo. “Hybrid Planning Heuristics Based on Task Decomposition Graphs”. In: *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS 2014)*. AAAI Press, 2014, pp. 35–43.

[8] **P. Bercher**, T. Geier, and S. Biundo. “Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning”. In: *Advances in Artificial Intelligence, Proceedings of the 36th German Conference on Artificial Intelligence (KI 2013)*. Springer, 2013, pp. 1–12. DOI: 10.1007/978-3-642-40942-4_1.

[9] **P. Bercher**, T. Geier, F. Richter, and S. Biundo. “On Delete Relaxation in Partial-Order Causal-Link Planning”. In: *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*. IEEE Computer Society, 2013, pp. 674–681. DOI: 10.1109/ICTAI.2013.105.

[10] M. Elkawkagy, **P. Bercher**, B. Schattenberg, and S. Biundo. “Improving Hierarchical Planning Performance by the Use of Landmarks”. In: *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012, pp. 1763–1769.

[11] S. Biundo, **P. Bercher**, T. Geier, F. Müller, and B. Schattenberg. “Advanced user assistance based on AI planning”. In: *Cognitive Systems Research* 12.3-4 (Apr. 2011). Special Issue on Complex Cognition, pp. 219–236. DOI: 10.1016/j.cogsys.2010.12.005.

[12] T. Geier and **P. Bercher**. “On the Decidability of HTN Planning with Task Insertion”. In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. AAAI Press, 2011, pp. 1955–1961.

6.2 Related Work

The related work is divided in two separate lists. The first one lists publications that were done by the author of the thesis, and the second one lists publications that were not – i.e., related work from the literature.

Related Work by the Author

The related work from the author of this thesis is marked by a preceding "A". Note that only publications are listed (i.e., cited) that are closely related to the thesis.

[A1] G. Behnke, B. Leichtmann, **P. Bercher**, D. Höller, V. Nitsch, M. Baumann, and S. Biundo. "Help me make a dinner! Challenges when assisting humans in action planning". In: *Proceedings of the 2nd International Conference on Companion Technology (ICCT 2017)*. IEEE, 2017.

[A2] G. Behnke, F. Nielsen, M. Schiller, **P. Bercher**, M. Kraus, W. Minker, B. Glimm, and S. Biundo. "SLOTH – the Interactive Workout Planner". In: *Proceedings of the 2nd International Conference on Companion Technology (ICCT 2017)*. IEEE, 2017.

[A3] G. Behnke, F. Nielsen, M. Schiller, D. Ponomaryov, **P. Bercher**, B. Glimm, W. Minker, and S. Biundo. "To Plan for the User Is to Plan With the User – Integrating User Interaction Into the Planning Process". In: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by S. Biundo and A. Wendemuth. Cognitive Technologies. In print. Springer, 2017. Chap. 7, pp. 123–144. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4_7.

[A4] F. Nothdurft, **P. Bercher**, G. Behnke, and W. Minker. "User Involvement in Collaborative Decision-Making Dialog Systems". In: *Dialogues with Social Robots: Enablements, Analyses, and Evaluation*. Ed. by K. Jokinen and G. Wilcock. This book chapter was accepted at the 7th International Workshop On Spoken Dialogue Systems (IWSDS 2016). Springer, 2017, pp. 129–141. DOI: 10.1007/978-981-10-2585-3_10.

[A5] **P. Bercher**, F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nielsen, F. Honold, F. Schüssel, S. Reuter, W. Minker, M. Weber, K. Dietmayer, and S. Biundo. "Advanced User Assistance for Setting Up a Home Theater". In: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by S. Biundo and A. Wendemuth. Cognitive Technologies. In print. Springer, 2017. Chap. 24, pp. 485–491. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4_24.

[A6] M. Schiller, G. Behnke, M. Schmautz, **P. Bercher**, M. Kraus, W. Minker, B. Glimm, and S. Biundo. “A Paradigm for Coupling Procedural and Conceptual Knowledge in Companion Systems”. In: *Proceedings of the 2nd International Conference on Companion Technology (ICCT 2017).* An extended version with additional authors will replace the paper referenced here. IEEE, 2017.

[A7] R. Alford, G. Behnke, D. Höller, **P. Bercher**, S. Biundo, and D. Aha. “Bound to Plan: Exploiting Classical Heuristics via Automatic Translations of Tail-Recursive HTN Problems”. In: *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016).* AAAI Press, 2016, pp. 20–28.

[A8] G. Behnke, D. Höller, **P. Bercher**, and S. Biundo. “Change the Plan – How hard can that be?” In: *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016).* AAAI Press, 2016, pp. 38–46.

[A9] S. Biundo, D. Höller, and **P. Bercher**. “Special Issue on Companion Technologies”. In: *Künstliche Intelligenz* 30.1 (2016). (Editorial), pp. 5–9. DOI: 10.1007/s13218-015-0421-9.

[A10] D. Höller, G. Behnke, **P. Bercher**, and S. Biundo. “Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages”. In: *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS 2016).* AAAI Press, 2016, pp. 158–165.

[A11] **P. Bercher** and D. Höller. “Interview with David E. Smith”. In: *Künstliche Intelligenz* 30.1 (2016). Special Issue on Companion Technologies, pp. 101–105. DOI: 10.1007/s13218-015-0403-y.

[A12] R. Alford, **P. Bercher**, and D. Aha. “Tight Bounds for HTN Planning”. In: *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015).* AAAI Press, 2015, pp. 7–15.

[A13] R. Alford, **P. Bercher**, and D. Aha. “Tight Bounds for HTN planning with Task Insertion”. In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015).* AAAI Press, 2015, pp. 1502–1508.

[A14] R. Alford, **P. Bercher**, and D. Aha. “Tight Bounds for HTN planning with Task Insertion (Extended Abstract)”. In: *Proceedings of the 8th Annual Symposium on Combinatorial Search (SoCS 2015).* This is an extended abstract of the paper [A13]. AAAI Press, 2015, pp. 221–222.

[A15] G. Behnke, **P. Bercher**, S. Biundo, B. Glimm, D. Ponomaryov, and M. Schiller. “Integrating Ontologies and Planning for Cognitive Systems”. In: *Proceedings of the 28th International Workshop on Description Logics (DL 2015).* CEUR Workshop Proceedings, 2015.

[A16] G. Behnke, D. Ponomaryov, M. Schiller, **P. Bercher**, F. Nothdurft, B. Glimm, and S. Biundo. “Coherence Across Components in Cognitive Systems – One Ontology to Rule Them All”. In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2015).* AAAI Press, 2015, pp. 1442–1449.

[A17] **P. Bercher**. "Hybrid Planning – Theoretical Foundations and Practical Applications". In: *Doctoral Consortium at ICAPS 2015 (ICAPS DC 2015)*. 2015.

[A18] F. Nothdurft, G. Behnke, **P. Bercher**, S. Biundo, and W. Minker. "The Interplay of User-Centered Dialog Systems and AI Planning". In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL 2015)*. Association for Computational Linguistics, 2015, pp. 344–353.

[A19] **P. Bercher**, D. Höller, G. Behnke, and S. Biundo. "User-Centered Planning – A Discussion on Planning in the Presence of Human Users". In: *Proceedings of the International Symposium on Companion Technology (ISCT 2015)*. 2015, pp. 79–83.

[A20] D. Höller, G. Behnke, **P. Bercher**, and S. Biundo. "Language Classification of Hierarchical Planning Problems". In: *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. IOS Press, 2014, pp. 447–452. DOI: 10.3233/978-1-61499-419-0-447.

[A21] D. Höller, **P. Bercher**, F. Richter, M. Schiller, T. Geier, and S. Biundo. "Finding User-friendly Linearizations of Partially Ordered Plans". In: *28th PuK Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK 2014)*. 2014.

[A22] F. Honold, **P. Bercher**, F. Richter, F. Nothdurft, T. Geier, R. Barth, T. Hörnle, F. Schüssel, S. Reuter, M. Rau, G. Bertrand, B. Seegebarth, P. Kurzok, B. Schattenberg, W. Minker, M. Weber, and S. Biundo. "Companion-Technology: Towards User- and Situation-Adaptive Functionality of Technical Systems". In: *10th International Conference on Intelligent Environments (IE 2014)*. IEEE, 2014, pp. 378–381. DOI: 10.1109/IE.2014.60.

[A23] **P. Bercher** and S. Biundo. "Encoding Partial Plans for Heuristic Search". In: *Proceedings of the 4th Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2013)*. 2013, pp. 11–15.

[A24] **P. Bercher** and S. Biundo. "A Heuristic for Hybrid Planning with Preferences". In: *Proceedings of the 25th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2012)*. AAAI Press, 2012, pp. 120–123.

[A25] R. Mattmüller, M. Ortlieb, M. Helmert, and **P. Bercher**. "Pattern Database Heuristics for Fully Observable Nondeterministic Planning". In: *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*. AAAI Press, 2010, pp. 105–112.

[A26] **P. Bercher** and R. Mattmüller. "Solving Non-deterministic Planning Problems with Pattern Database Heuristics". In: *Advances in Artificial Intelligence, Proceedings of the 32nd German Conference on Artificial Intelligence (KI 2009)*. Springer, 2009, pp. 57–64. DOI: 10.1007/978-3-642-04617-9_8.

Related Work from the Literature

The related work from the literature is marked by a preceding "L".

[L1] Retrieved on Jule, 29th, 2017. URL: http://sig-aps.org/.

[L2] G. Behnke, D. Höller, and S. Biundo. "This is a solution! (... but is it though?) – Verifying solutions of hierarchical planning problems". In: *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS 2017)*. AAAI Press, 2017, pp. 20–28.

[L3] S. Biundo and A. Wendemuth, eds. *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Cognitive Technologies. In print. Springer, 2017. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4.

[L4] F. Richter. "Hierarchical Planning Under Uncertainty". PhD thesis. Ulm University, Germany, 2017.

[L5] F. Richter and S. Biundo. "Addressing Uncertainty in Hierarchical User-Centered Planning". In: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by S. Biundo and A. Wendemuth. Cognitive Technologies. In print. Springer, 2017. Chap. 6, pp. 101–121. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4_6.

[L6] V. Shivashankar, R. Alford, and D. Aha. "Incorporating Domain-Independent Planning Heuristics in Hierarchical Planning". In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017)*. AAAI Press, 2017, pp. 3658–3664.

[L7] Z. Xiao, A. Herzig, L. Perrussel, H. Wan, and X. Su. "Hierarchical Task Network Planning with Task Insertion and State Constraints". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*. AAAI Press, 2017, pp. 4463–4469. DOI: 10.24963/ijcai.2017/623.

[L8] R. Alford, V. Shivashankar, M. Roberts, J. Frank, and D. W. Aha. "Hierarchical planning: relating task and goal decomposition with task sharing". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*. AAAI Press, 2016, pp. 3022–3029.

[L9] S. Biundo and A. Wendemuth. "*Companion*-Technology for Cognitive Technical Systems". In: *Künstliche Intelligenz* 30.1 (2016). Special Issue on Companion Technologies, pp. 71–75. DOI: 10.1007/s13218-015-0414-8.

[L10] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN: 978-1-107-03727-4.

[L11] D. Lotinac and A. Jonsson. "Constructing Hierarchical Task Models Using Invariance Analysis". In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. IOS Press, 2016, pp. 1274–1282. DOI: 10.3233/978-1-61499-672-9-1274.

[L12] F. M. Nothdurft. "User- and Situation-adaptive Explanations in Dialogue Systems". PhD thesis. Ulm University, Germany, 2016.

[L13] V. Shivashankar, R. Alford, M. Roberts, and D. W. Aha. "Cost-Optimal Algorithms for Planning with Procedural Control Knowledge". In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. IOS Press, 2016, pp. 1702–1703.

[L14] G. Behnke, D. Höller, and S. Biundo. "On the Complexity of HTN Plan Verification and its Implications for Plan Recognition". In: *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS 2015)*. AAAI Press, 2015, pp. 25–33.

[L15] I. Georgievski and M. Aiello. "HTN planning: Overview, comparison, and beyond". In: *Artificial Intelligence* 222 (2015), pp. 124–156. DOI: 10.1016/j.artint.2015.02.002.

[L16] A. Jonsson and D. Lotinac. "Automatic Generation of HTNs From PDDL". In: *Proceedings of the 5th workshop on Planning and Learning (PAL 2015)*. 2015.

[L17] V. Shivashankar. "Hierarchical Goal Network: Formalisms and Algorithms for Planning and Acting". PhD thesis. University of Maryland, 2015.

[L18] R. Alford, V. Shivashankar, U. Kuter, and D. Nau. "On the Feasibility of Planning Graph Style Heuristics for HTN Planning". In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*. AAAI Press, 2014, pp. 2–10.

[L19] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal. "HiPOP: Hierarchical Partial-Order Planning". In: *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS 2014)*. IOS Press, 2014, pp. 51–60. DOI: 10.3233/978-1-61499-421-3-51.

[L20] F. Dvorák, A. Bit-Monnot, F. Ingrand, and M. Ghallab. "A Flexible ANML Actor and Planner in Robotics". In: *Proceedings of the 2nd Workshop on Planning and Robotics (PlanRob 2014)*. 2014, pp. 12–19.

[L21] R. Alford. "Search Complexities for HTN Planning". PhD thesis. University of Maryland, 2013.

[L22] S. Bernardini and K. Porayska-Pomsta. "Planning-Based Social Partners for Children with Autism". In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*. AAAI Press, 2013, pp. 362–370.

[L23] F. Honold, F. Schüssel, M. Weber, F. Nothdurft, G. Bertrand, and W. Minker. "Context Models for Adaptive Dialogs and Multimodal Interaction". In: *Proceedings of the 9th International Conference on Intelligent Environments (IE 2013)*. IEEE, 2013, pp. 57–64. DOI: 10.1109/IE.2013.54.

[L24] R. Petrick and M. E. Foster. “Planning for Social Interaction in a Robot Bartender Domain”. In: *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*. AAAI Press, 2013, pp. 389–397.

[L25] V. Shivashankar, R. Alford, U. Kuter, and D. Nau. “The GoDeL Planning System: A More Perfect Union of Domain-Independent and Hierarchical Planning”. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. AAAI Press, 2013, pp. 2380–2386.

[L26] R. Alford, V. Shivashankar, U. Kuter, and D. S. Nau. “HTN Problem Spaces: Structure, Algorithms, Termination”. In: *Proceedings of The 5th Annual Symposium on Combinatorial Search (SoCS 2012)*. AAAI Press, 2012, pp. 2–9.

[L27] M. Beetz, D. Jain, L. Mösenlechner, M. Tenorth, L. Kunze, N. Blodow, and D. Pangercic. “Cognition-Enabled Autonomous Robot Control for the Realization of Home Chore Task Intelligence”. In: *Proceedings of the IEEE* 100.8 (2012), pp. 2454–2471. DOI: 10.1109/JPROC.2012.220055.

[L28] B. Seegebarth, F. Müller, B. Schattenberg, and S. Biundo. “Making Hybrid Plans More Clear to Human Users – A Formal Approach for Generating Sound Explanations”. In: *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press, 2012, pp. 225–233.

[L29] V. Shivashankar, U. Kuter, D. Nau, and R. Alford. “A hierarchical goal-based formalism and algorithm for single-agent planning”. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*. International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 981–988.

[L30] F. H. Siddiqui and P. Haslum. “Block-Structured Plan Deordering”. In: *Advances in Artificial Intelligence: 25th Australasian Joint Conference (AI 2012)*. Springer, 2012, pp. 803–814. DOI: 10.1007/978-3-642-35101-3_68.

[L31] E. Karpas and C. Domshlak. “Living on the Edge: Safe Search with Unsafe Heuristics”. In: *Proceedings of the 2011 ICAPS Workshop on Heuristics for Domain Independent Planning (HDIP)*. 2011.

[L32] M. Buss and M. Beetz. “CoTeSys – Cognition for Technical Systems”. In: *Künstliche Intelligenz* 24.4 (2010), pp. 323–327. DOI: 10.1007/s13218-010-0061-z.

[L33] M. Elkawkagy, B. Schattenberg, and S. Biundo. “Landmarks in Hierarchical Planning”. In: *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2010)*. IOS Press, 2010, pp. 229–234. DOI: 10.3233/978-1-60750-606-5-229.

[L34] H. J. Ritter. “Cognitive Interaction Technology – Goals and Perspectives of Excellence Cluster CITEC”. In: *Künstliche Intelligenz* 24.4 (2010), pp. 319–322. DOI: 10.1007/s13218-010-0063-x.

[L35] R. Alford, U. Kuter, and D. S. Nau. “Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*. AAAI Press, 2009, pp. 1629–1634.

[L36] M. Helmert and C. Domshlak. “Landmarks, Critical Paths and Abstractions: What’s the Difference Anyway?” In: *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI Press, 2009, pp. 162–169.

[L37] M. Ramírez and H. Geffner. “Plan Recognition as Planning”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*. AAAI Press, 2009, pp. 1778–1783.

[L38] B. Schattenberg. “Hybrid Planning & Scheduling”. PhD thesis. University of Ulm, Germany, 2009. DOI: `10.18725/OPARU-1045`.

[L39] S. Sohrabi, J. A. Baier, and S. A. McIlraith. “HTN Planning with Preferences”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*. AAAI Press, 2009, pp. 1790–1797.

[L40] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. 2009. URL: `http://www.w3.org/TR/owl2-overview/`.

[L41] J. Bidot, B. Schattenberg, and S. Biundo. “Plan Repair in Hybrid Planning”. In: *Advances in Artificial Intelligence, Proceedings of the 31st German Conference on Artificial Intelligence (KI 2008)*. Springer, 2008, pp. 169–176. DOI: `10.1007/978-3-540-85845-4_21`.

[L42] A. Gerevini, U. Kuter, D. S. Nau, A. Saetti, and N. Waisbrot. “Combining Domain-Independent Planning and HTN Planning: The Duet Planner”. In: *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*. IOS Press, 2008, pp. 573–577. DOI: `10.3233/978-1-58603-891-5-573`.

[L43] N. Lin, U. Kuter, and E. Sirin. “Web Service Composition with User Preferences”. In: *ESWC’08: Proceedings of the 5th European Semantic Web Conference*. Springer, 2008, pp. 629–643. DOI: `10.1007/978-3-540-68234-9_46`.

[L44] B. Marthi, S. Russell, and J. Wolfe. “Angelic Hierarchical Planning: Optimal and Online Algorithms”. In: *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*. AAAI Press, 2008, pp. 222–231.

[L45] D. E. Smith, J. Frank, and W. Cushing. “The ANML language”. In: *In Proc. of the Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2008)*. 2008.

[L46] M. Beetz, M. Buss, and D. Wollherr. “Cognitive Technical Systems – What Is the Role of Artificial Intelligence?” In: *Proceedings of the 30th German Conference on Artificial Intelligence (KI 2013)*. Springer, 2007, pp. 19–42. DOI: `10.1007/978-3-540-74565-5_3`.

[L47] M. Helmert, P. Haslum, and J. Hoffmann. “Flexible Abstraction Heuristics for Optimal Sequential Planning”. In: *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007).* AAAI Press, 2007, pp. 176–183.

[L48] B. Marthi, S. J. Russell, and J. Wolfe. “Angelic Semantics for High-Level Actions”. In: *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007).* AAAI Press, 2007, pp. 232–239.

[L49] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, and A. Mihailidis. “A planning system based on Markov decision processes to guide people with dementia through activities of daily living”. In: *IEEE Transactions on Information Technology in Biomedicine* 10.2 (2006), pp. 323–333. DOI: 10.1109/TITB.2006.864480.

[L50] P. Haslum. “Admissible Heuristics for Automated Planning”. PhD thesis. Department of Computer and Information Science, Linköpings Universitet, 2006.

[L51] M. Helmert. “The Fast Downward Planning System”. In: *Journal of Artificial Intelligence Research (JAIR)* 26 (2006), pp. 191–246.

[L52] V. Vidal and H. Geffner. “Branching and pruning: An optimal temporal POCL planner based on constraint programming”. In: *Artificial Intelligence* 170 (2006), pp. 298–335. DOI: 10.1016/j.artint.2005.08.004.

[L53] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis. “A decision-theoretic approach to task assistance for persons with dementia”. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005).* Professional Book Center, 2005, pp. 1293–1299.

[L54] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Muñoz-Avila, and J. W. Murdock. “Applications of SHOP and SHOP2”. In: *Intelligent Systems, IEEE* 20 (Mar. 2005), pp. 34–41. DOI: 10.1109/MIS.2005.20.

[L55] M. Ghallab, D. S. Nau, and P. Traverso. *Automated Planning: Theory and Practice.* Ed. by D. E. M. Penrose. Morgan Kaufmann, 2004. ISBN: 1558608567.

[L56] M. Ghallab, D. S. Nau, and P. Traverso. “Hierarchical Task Network Planning”. In: *Automated Planning: Theory and Practice.* Ed. by D. E. M. Penrose. Morgan Kaufmann, 2004. Chap. 11, pp. 229–261. ISBN: 1558608567.

[L57] M. Ghallab, D. S. Nau, and P. Traverso. “Plan-Space Planning”. In: *Automated Planning: Theory and Practice.* Ed. by D. E. M. Penrose. Morgan Kaufmann, 2004. Chap. 5, pp. 85–109. ISBN: 1558608567.

[L58] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. “SHOP2: An HTN Planning System”. In: *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), pp. 379–404.

[L59] H. L. S. Younes and R. G. Simmons. “VHPOP: Versatile heuristic partial order planner”. In: *Journal of Artificial Intelligence Research (JAIR)* 20 (2003), pp. 405–430.

[L60] M. E. Pollack. “Planning Technology for Intelligent Cognitive Orthotics”. In: *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*. AAAI Press, 2002, pp. 322–332.

[L61] S. Biundo and B. Schattenberg. “From Abstract Crisis to Concrete Relief – A Preliminary Report on Combining State Abstraction and HTN Planning”. In: *Proceedings of the 6th European Conference on Planning (ECP 2001)*. AAAI Press, 2001, pp. 157–168.

[L62] L. A. Castillo, J. Fernández-Olivares, and A. González. “On the Adequacy of Hierarchical Planning Characteristics for Real-World Problem Solving”. In: *Proceedings of the 6th European Conference on Planning (ECP 2001)*. AAAI Press, 2001, pp. 169–180.

[L63] J. Hoffmann and B. Nebel. “The FF Planning System: Fast Plan Generation Through Heuristic Search”. In: *Journal of Artificial Intelligence Research (JAIR)* 14 (May 2001), pp. 253–302.

[L64] X. Nguyen and S. Kambhampati. “Reviving Partial Order Planning”. In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*. Morgan Kaufmann, 2001, pp. 459–466.

[L65] P. Haslum and H. Geffner. “Admissible Heuristics for Optimal Planning”. In: *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000)*. AAAI Press, 2000, pp. 140–149.

[L66] D. McDermott. “The 1998 AI Planning Systems Competition”. In: *AI Magazine* 21.2 (2000), pp. 35–55.

[L67] A. Lotem, D. S. Nau, and J. A. Hendler. “Using planning graphs for solving HTN planning problems”. In: *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 1999)*. AAAI Press, 1999, pp. 534–540.

[L68] S. Kambhampati, A. Mali, and B. Srivastava. “Hybrid Planning for Partially Hierarchical Domains”. In: *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*. AAAI Press, 1998, pp. 882–888.

[L69] A. L. Blum and M. L. Furst. “Fast Planning Through Planning Graph Analysis”. In: *Artificial Intelligence* 90 (1997), pp. 281–300. DOI: 10.1016/S0004-3702(96)00047-1.

[L70] B. Bonet, G. Loerincs, and H. Geffner. “A Robust and Fast Action Selection Mechanism for Planning”. In: *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997)*. AAAI Press, 1997, pp. 8–14.

[L71] T. A. Estlin, S. A. Chien, and X. Wang. “An Argument for a Hybrid HTN/Operator-Based Approach to Planning”. In: *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning (ECP 1997)*. 1997, pp. 182–194.

[L72] M. Fox. "Natural Hierarchical Planning using Operator Decomposition". In: *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning (ECP 1997)*. Springer, 1997, pp. 195–207. DOI: 10.1007/3-540-63912-8_86.

[L73] B. C. Gazen and C. A. Knoblock. "Combining the Expressivity of UCPOP with the Efficiency of Graphplan". In: *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning (ECP 1997)*. Springer, 1997, pp. 221–233. DOI: 10.1007/3-540-63912-8_88.

[L74] M. E. Pollack, D. Joslin, and M. Paolucci. "Flaw Selection Strategies For Partial-Order Planning". In: *Journal of Artificial Intelligence Research (JAIR)* 6 (1997), pp. 223–262.

[L75] K. Erol. "Hierarchical Task Network Planning: Formalization, Analysis, and Implementation". PhD thesis. University of Maryland, 1996.

[L76] K. Erol, J. A. Hendler, and D. S. Nau. "Complexity results for HTN planning". In: *Annals of Mathematics and Artificial Intelligence* 18.1 (1996), pp. 69–93. DOI: 10.1007/BF02136175.

[L77] M. Williamson and S. Hanks. "Flaw Selection Strategies for Value-Directed Planning". In: *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS 1996)*. AAAI Press, 1996, pp. 237–244.

[L78] C. Bäckström and B. Nebel. "Complexity Results for SAS+ Planning". In: *Computational Intelligence* 11.4 (1995), pp. 626–655. DOI: 10.1111/j.1467-8640.1995.tb00052.x.

[L79] M. Fox and D. Long. "Hierarchical planning using abstraction". In: *IEE Proc. – Control Theory Appl.* 142.3 (1995), pp. 197–210. DOI: 10.1049/ip-cta:19951848.

[L80] S. Kambhampati. "Admissible Pruning Strategies based on plan minimality for Plan-Space Planning". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*. Morgan Kaufmann, Aug. 1995, pp. 1627–1633.

[L81] T. Bylander. "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 94.1-2 (1994), pp. 165–204. DOI: 10.1016/0004-3702(94)90081-7.

[L82] S. Minton, J. Bresina, and M. Drummond. "Total-Order and Partial-Order Planning: A Comparative Analysis". In: *Journal of Artificial Intelligence Research (JAIR)* 2 (1994), pp. 227–262.

[L83] B. M. Muir. "Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems". In: *Ergonomics* 37.11 (1994), pp. 1905–1922. DOI: 10.1080/00140139408964957.

[L84] B. Nebel and C. Bäckström. "On the Computational Complexity of Temporal Projection, Planning, and Plan Validation". In: *Artificial Intelligence* 66.1 (1994), pp. 125–160. DOI: 10.1016/0004-3702(94)90005-1.

[L85] S. Russell and P. Norvig. In: *Artificial Intelligence – A modern Approach.* 1st ed. Prentice-Hall, 1994. Chap. 12: Practical Planning, pp. 367–376. ISBN: 0133601242.

[L86] R. M. Young, M. E. Pollack, and J. D. Moore. "Decomposition and Causality in Partial-Order Planning". In: *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994).* AAAI Press, 1994, pp. 188–193.

[L87] S. Kambhampati and J. A. Hendler. "A validation-structure-based theory of plan modification and reuse". In: *Artificial Intelligence* 55 (1992), pp. 193–258. DOI: 10.1016/0004-3702(92)90056-4.

[L88] J. S. Penberthy and D. S. Weld. "UCPOP: A Sound, Complete, Partial Order Planner for ADL". In: *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR 1992).* Morgan Kaufmann, 1992, pp. 103–114.

[L89] D. McAllester and D. Rosenblitt. "Systematic Nonlinear Planning". In: *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI 1991).* AAAI Press, 1991, pp. 634–639.

[L90] Q. Yang. "Formalizing Planning Knowledge for Hierarchical Planning". In: *Computational Intelligence* 6.1 (1990), pp. 12–24. DOI: 10.1111/j.1467-8640.1990.tb00126.x.

[L91] D. Chapman. "Planning for Conjunctive Goals". In: *Artificial Intelligence* 32.3 (July 1987), pp. 333–377. DOI: 10.1016/0004-3702(87)90092-0.

[L92] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979. ISBN: 0-201-02988-X.

[L93] E. D. Sacerdoti. "The Nonlinear Nature of Plans". In: *Proceedings of the 4th International Joint Conference on Artificial Intelligence (IJCAI 1975).* Morgan Kaufmann Publishers Inc., 1975, pp. 206–214.

[L94] R. E. Fikes and N. J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2 (1971), pp. 189–208.

[L95] G. Pólya. *How to Solve It.* Princeton University Press, 1945. ISBN: 0-691-08097-6.

[L96] A. de Saint-Exupéry. *Wind, Sand and Stars.* Reynal & Hitchcock, 1939.

7 Core Contributions in Full Length

This section includes the publications of the thesis' core contributions in full length. The reference numbers are the same as in the preceding of the thesis. We cluster the publications according to the sections in which they have been described.

7.1 Theoretical Foundations

[4] **P. Bercher**, D. Höller, G. Behnke, and S. Biundo. "More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks". In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*. IOS Press, 2016, pp. 225–233. DOI: 10.3233/978-1-61499-672-9-225.

....94 – 103

[9] **P. Bercher**, T. Geier, F. Richter, and S. Biundo. "On Delete Relaxation in Partial-Order Causal-Link Planning". In: *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*. IEEE Computer Society, 2013, pp. 674–681. DOI: 10.1109/ICTAI.2013.105.

....104 – 112

[12] T. Geier and **P. Bercher**. "On the Decidability of HTN Planning with Task Insertion". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. AAAI Press, 2011, pp. 1955–1961.

....114 – 121

7.2 Search and Heuristics

[1] **P. Bercher**, G. Behnke, D. Höller, and S. Biundo. “An Admissible HTN Planning Heuristic”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*. AAAI Press, 2017, pp. 480–488. DOI: 10.24963/ijcai.2017/68.

. . . . 122 – 131

[7] **P. Bercher**, S. Keen, and S. Biundo. “Hybrid Planning Heuristics Based on Task Decomposition Graphs”. In: *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS 2014)*. AAAI Press, 2014, pp. 35–43.

. . . . 132 – 141

[8] **P. Bercher**, T. Geier, and S. Biundo. “Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning”. In: *Advances in Artificial Intelligence, Proceedings of the 36th German Conference on Artificial Intelligence (KI 2013)*. Springer, 2013, pp. 1–12. DOI: 10.1007/978-3-642-40942-4_1.

. . . . 142 – 154

[9] **P. Bercher**, T. Geier, F. Richter, and S. Biundo. “On Delete Relaxation in Partial-Order Causal-Link Planning”. In: *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*. IEEE Computer Society, 2013, pp. 674–681. DOI: 10.1109/ICTAI.2013.105.

. . . . 104 – 112

[10] M. Elkawkagy, **P. Bercher**, B. Schattenberg, and S. Biundo. “Improving Hierarchical Planning Performance by the Use of Landmarks”. In: *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012, pp. 1763–1769.

. . . . 156 – 163

7.3 Practical Application

[2] **P. Bercher**, D. Höller, G. Behnke, and S. Biundo. “User-Centered Planning”. In: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by S. Biundo and A. Wendemuth. Cognitive Technologies. In print. Springer, 2017. Chap. 5, pp. 79–100. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4_5.

. . . . 164 – 185

[3] S. Biundo, D. Höller, B. Schattenberg, and **P. Bercher**. “Companion-Technology: An Overview”. In: *Künstliche Intelligenz* 30.1 (2016). Special Issue on Companion Technologies, pp. 11–20. DOI: 10.1007/s13218-015-0419-3.

. . . . 186 – 197

[5] **P. Bercher**, F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nothdurft, F. Honold, W. Minker, M. Weber, and S. Biundo. “A Planning-based Assistance System for Setting Up a Home Theater”. In: *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, 2015, pp. 4264–4265.

. . . . 198 – 200

[6] **P. Bercher**, S. Biundo, T. Geier, T. Hörnle, F. Nothdurft, F. Richter, and B. Schattenberg. “Plan, Repair, Execute, Explain - How Planning Helps to Assemble your Home Theater”. In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*. AAAI Press, 2014, pp. 386–394.

. . . . 202 – 211

[11] S. Biundo, **P. Bercher**, T. Geier, F. Müller, and B. Schattenberg. “Advanced user assistance based on AI planning”. In: *Cognitive Systems Research* 12.3-4 (Apr. 2011). Special Issue on Complex Cognition, pp. 219–236. DOI: 10.1016/j.cogsys.2010.12.005.

. . . . 212 – 246

The following pages show the publication:

P. Bercher, D. Höller, G. Behnke, and S. Biundo. “More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks”. In: *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016).* IOS Press, 2016, pp. 225–233. DOI: 10.3233/978-1-61499-672-9-225

The included PDF is a revised version. The modifications are noted in the acknowledgements.

More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks

Pascal Bercher and **Daniel Höller** and **Gregor Behnke** and **Susanne Biundo** [*]

Abstract. There are several formalizations for hierarchical planning. Many of them allow to specify preconditions and effects for compound tasks. They can be used, e.g., to assist during the modeling process by ensuring that the decomposition methods' plans "implement" the compound tasks' intended meaning. This is done based on so-called *legality criteria* that relate these preconditions and effects to the method's plans and pose further restrictions. Despite the variety of expressive hierarchical planning formalisms, most theoretical investigations are only known for standard HTN planning, where compound tasks are just names, i.e., no preconditions or effects can be specified. Thus, up to now, a direct comparison to other hierarchical planning formalisms is hardly possible and fundamental theoretical properties are yet unknown. To enable a better comparison between such formalisms (in particular with respect to their computational expressivity), we first provide a survey on the different legality criteria known from the literature. Then, we investigate the theoretical impact of these criteria for two fundamental problems to planning: *plan verification* and *plan existence*. We prove that the plan verification problem is at most $\mathbf{NP}$-$\mathbf{complete}$, while the plan existence problem is in the general case both $\mathbf{semi\text{-}decidable}$ and $\mathbf{undecidable}$, independent of the demanded criteria. Finally, we discuss our theoretical findings and practical implications.

1 Introduction

Hierarchical planning approaches are often chosen when it comes to practical real-world planning applications [33]. Examples include composition of web services [29], real-time strategy games [23, 35], robotics [15, 28], or user assistance [11, 10]. While there are several different formalizations for hierarchical planning, it is apparent that most of the *theoretical* investigations are done for a standard formalization (called hierarchical task network (HTN) planning), where compound (or abstract) tasks are just names or symbols [16, 19] – they thus neither show preconditions nor effects. Those investigations include the complexity of the plan existence problem ("Is the problem solvable?") [16, 19, 5, 2, 3], plan verification ("Is the given plan a solution to the problem?") [9], changes to plans ("Is the plan still a solution if I change X?") [8], and expressivity analysis ("What plan structures can be expressed using different language features?") [21, 22]. The answers to such questions, besides being of theoretical interest, are highly relevant to come up with tractable problem relaxations for heuristics [5] or for problem compilations [4, 1].

For mainly practically motivated reasons, such as providing modeling assistance or generating abstract solutions, several researchers developed hierarchical planning formalizations in which compound tasks are allowed to have preconditions and effects [40, 25, 37, 41, 18, 26, 12, 14, 30, 11, 7, 15] (cf. example given in Fig. 1). However, the only theoretical investigations for such a formalization that we are aware of are about the upward and downward refinement properties [40, 6, 30]. So, up to now, for many of such formalisms it is not even clear how hard the respective problems are (given the typical HTN solution criteria), since the plan existence problem was not studied for such a formalization. To close this gap, we investigate the plan existence and the plan verification problems for our formalization that allows to specify preconditions and effects for compound tasks. We survey several legality criteria that define which decomposition methods may be specified for which compound task, depending on its preconditions and effects and take these criteria into account in our complexity analysis. We conclude the paper by discussing our findings and its implications.

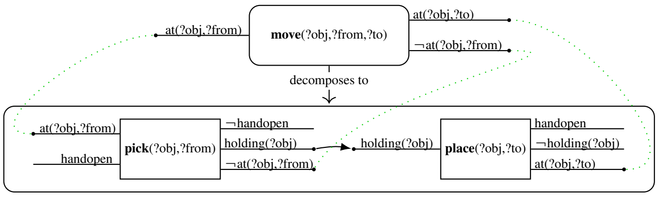


Figure 1. Example for a compound task (*move*) with preconditions and effects and one of its methods. The method's plan consists of two primitive tasks (*pick* and *place*) and a causal link protecting the condition *holding*. The dotted green lines indicate how the preconditions and effects of the tasks in the plan's method are related to their more abstract representation.

2 Problem Formalization

We introduce a hierarchical planning formalism that allows to specify preconditions and effects for compound tasks. In our complexity analysis, we take into account several restrictions on the relationship between compound tasks and the plans that are associated with them via their decomposition methods. The investigated restrictions, called legality criteria, are taken from the literature. To formally study their impact on the complexity results, the formalism needs to be rich enough to be capable of expressing these criteria. Most of the formalisms that define them [40, 41, 37, 12] fuse standard HTN planning [16, 20, 19] with Partial-Order Causal-Link (POCL) planning [31, 36, 20]. We assume this is because the concept of causal links makes it easy to express the desired criteria. Thus, we also use such an hybridization for our investigations. A variety of formalisms fuse HTN with POCL planning [40, 25, 37, 41, 18, 14, 7, 15, 11].

[*] Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany, {*forename.surname*}@uni-ulm.de

However, none of these formalizations is both rich enough to allow expressing all legality criteria while being simple enough to easily serve as a basis for proofs. Since all HTN complexity results that are relevant for the sake of this paper have been shown or reproduced in the simplistic (propositional) HTN formalism by Geier and Bercher [19], extending it allows an easy comparison. We therefore extend it by the necessary POCL concepts. In accordance to the literature [26, 11], we refer to the resulting formalism as *hybrid planning*.

Let V be a finite set of *state variables* (or *proposition symbols*). In POCL planning, actions are typically 2-tuples consisting of a precondition and effect, both being conjunctions of literals. Here, we use an equivalent set-based formalization: Actions (or *primitive tasks*) are 4-tuples $(\mathit{prec}^+, \mathit{prec}^-, \mathit{eff}^+, \mathit{eff}^-)$, where prec^+ and prec^- denote the positive and negative *preconditions* and eff^+ and eff^- denote the positive and negative *effects*. They describe single state transitions as usual. The *compound* (or *abstract*) tasks have a different underlying meaning, as they need to be decomposed into predefined plans by relying on so-called *decomposition methods*. Despite that fact, we allow compound tasks to use preconditions and effects as well (cf. Fig. 1 for an example). Every task has a *task name*. The set of names for the primitive tasks is given by N_p and those of the compound ones by N_c. We define $N := N_p \dot{\cup} N_c$. The mapping between task names and their actual tasks (i.e., 4-tuples) is established using the function $\delta : N \to (2^V)^4$. For convenience, we also write $\mathit{prec}^+(n)$, $\mathit{prec}^-(n)$, $\mathit{eff}^+(n)$, and $\mathit{eff}^-(n)$ to refer to n's positive and negative preconditions and effects, respectively[†]. We call a sequence of tasks $\delta(n_1), \dots, \delta(n_k)$ *executable in a state* $s_0 \in 2^V$ if and only if there is a corresponding sequence of states $s_0, \dots, s_k$, such that for all $1 \le i \le k$ holds $\mathit{prec}^+(n_i) \subseteq s_{i-1}$ and $\mathit{prec}^-(n_i) \cap s_{i-1} = \emptyset$ as well as $s_i = (s_{i-1} \setminus \mathit{eff}^-(n_i)) \cup \mathit{eff}^+(n_i)$. The state s_k is called *the state generated by* $\delta(n_1), \dots, \delta(n_k)$.

In non-linear planning approaches, *plans* are only partially ordered. For a set of ordering constraints $\prec$, we denote its transitive closure by $\prec^*$. To differentiate multiple occurrences of the same task within a plan, the partial order is defined over a set of so-called *plan steps* PS, which then map to the actual task name, $\alpha : PS \to N$. When we mention a *linearization* of the plan steps of a plan, we refer to a total order of PS that does not violate the partial order $\prec$. Plans may also contain so-called *causal links*. A causal link $(ps, v, ps') \in PS \times V \times PS$ indicates that the precondition v of the *consumer* plan step ps' is *supported* by the *producer* plan step ps. The condition v is also said to be *protected* by the causal link. This means, if v is a positive (resp. negative) precondition of ps', then no task with v as negative (resp. positive) effect is allowed to be ordered between ps and ps'.

Definition 1 (Plan)**.** *A* plan *P over a set of task names N is a 4-tuple $(PS, CL, \prec, \alpha)$, where:*

- *PS is a finite (possibly empty) set of plan steps,*
- *$CL \subseteq PS \times V \times PS$ is a set of causal links. If $(ps, v, ps') \in CL$, then $v \in \mathit{prec}^+(ps')$ and $v \in \mathit{eff}^+(ps)$ or $v \in \mathit{prec}^-(ps')$ and $v \in \mathit{eff}^-(ps)$. We also require that every precondition variable of all plan steps is protected by at most one causal link,*
- *$\prec \subseteq PS \times PS$ is a strict partial order. If $(ps, v, ps') \in CL$ with $\alpha(ps)$ and $\alpha(ps')$ being primitive, then $(ps, ps') \in \prec^{‡}$,*
- *$\alpha : PS \to N$ labels every plan step with its task name.*

$\mathcal{P}_N$ denotes the set of all plans over the task names N. Two plans are called isomorphic if they are identical except for plan step renaming.

Based on the concept of plans, we define *decomposition methods*. The set of all decomposition methods $M \subseteq N_c \times \mathcal{P}_N$ is given in the planning domain and defines how compound tasks can be decomposed. That is, a method $(n_c, P) \in M$ indicates that the compound task (name) n_c can be decomposed into the plan P.

Definition 2 (Hybrid Planning Problem)**.** *A* hybrid planning problem *is a 6-tuple $\pi = (V, N_c, N_p, \delta, M, P^i)$, where:*

- *V is a finite set of* state variables*,*
- *we require $N_c \cap N_p = \emptyset$ and define $N := N_c \cup N_p$, where:*
  - *N_c is a finite set of* compound task names*,*
  - *N_p is a finite set of* primitive task names*,*
  - *$\{\mathit{init}, \mathit{goal}\} \subseteq N_p$ denote two special primitive task names,*
- *$\delta : N \to (2^V)^4$ is a function mapping the task names to their preconditions and effects*[†][§]*,*
- *$M \subseteq N_c \times \mathcal{P}_{N \setminus \{\mathit{init}, \mathit{goal}\}}$ is a finite set of* (decomposition) methods*, and*
- *$P^i = (PS^i, CL^i, \prec^i, \alpha^i) \in \mathcal{P}_N$, is the* initial plan*. We require that there are plan steps $ps, ps' \in PS^i$ such that:*
  - *$\alpha^i(ps) = \mathit{init}$ and $\alpha^i(ps') = \mathit{goal}$, and*
  - *$ps \prec ps'$ and for all $ps'' \in PS^i$ with $\{ps''\} \cap \{ps, ps'\} = \emptyset$ holds $ps \prec ps'' \prec ps'$.*

The actual problem that one would like to have solved is given in terms of the initial plan P^i. As done in POCL planning, this plan contains two artificial actions that encode the initial state and the goal description, respectively[§]. Since hybrid planning is a hierarchical setting, P^i usually contains a set of compound tasks for which one needs to find an executable refinement. We added the specification of a goal description for practical reasons: It is especially interesting if one allows the arbitrary insertion of tasks into the plan apart from decomposing compound tasks [19] (not considered in this paper), since hybrid planning then directly captures both classical and POCL planning. Independent of whether task insertion is allowed or not, adding a goal description is not required from a purely theoretical point of view, since one can easily simulate it [19, Sec. 2].

In HTN planning, only those plans are regarded solutions that can be obtained from the initial plan by successively applying decomposition methods to compound tasks. We thus need to define how applying a method transforms one plan into another. Since the decomposed task might serve as a producer or consumer of causal links, we have to decide how such links will be passed down to sub tasks and whether this is mandatory or not (i. e., we could even allow that such links may be deleted upon decomposition). Adding a causal link to a compound task means to commit that the state variable of that link is protected for the complete sequence of states over which this link spans. Allowing to remove that link upon decomposition would remove this constraint and violate the refinement principle [24]. If

[†] To simplify upcoming definitions, we require that for all primitive tasks n_p, $\mathit{prec}^+(n_p) \cap \mathit{prec}^-(n_p) = \mathit{eff}^+(n_p) \cap \mathit{eff}^-(n_p) = \emptyset$.

[‡] As usual in POCL planning, any causal link between primitive tasks implies an ordering. If one of these tasks was compound, this would not be reasonable: Consider the example depicted in Fig. 1 and assume there is a causal link from *move*'s effect $\neg at$ to another task's precondition. If that link would imply an ordering, then after *move* was decomposed, the consumer task had to be ordered behind *place*, which is overly restrictive.

[§] The initial state and goal description are specified in terms of the task names *init* and *goal* and their tuple representation using δ. As usual in POCL planning, the action for *init* does not show a precondition and uses the initial state as effect and, analogously, the action for *goal* has no effects and uses the goal description as precondition.

causal links do have to be passed down to sub tasks, then the respective formalism satisfies the so-called *monotonic property* [27]. If this property does not hold, hierarchical planning systems cannot exploit such causal links to prune plans from the search space, as the constraint imposed by these links could disappear upon decomposition [14]. We hence require that causal links are not allowed to disappear upon decomposition. *How* causal links are passed down has yet to be decided – and different conventions exist [25, p. 204]. We follow the canonical approach by Yang [40] and pass down every causal link to each "compatible" sub task. In other words, if there is a link to a compound task's precondition/effect v, then for each precondition/effect v in its sub tasks, one successor plan is generated in which the link is passed down to the respective task. In case there is more than one matching sub task, it is not required that the causal link is duplicated to support all these tasks (or a sub set thereof), since the respective plan can be obtained via link insertions from the other plans.

The following definitions formally capture the decomposition of compound tasks and the inheritance of causal links. We first define two functions $in_P, out_P : PS \rightarrow 2^{CL}$ that return the set of incoming, respectively outcoming, causal links of a plan step $ps \in PS$ in a plan $P = (PS, CL, \prec, \alpha)$ as $in_P : ps \mapsto \{(ps', v, ps) \in CL \mid ps' \in PS\}$ and $out_P : ps \mapsto \{(ps, v, ps') \in CL \mid ps' \in PS\}$.

Definition 3 (Decomposition)**.** *A method $m = (n_c, P) \in M$ decomposes a plan $P' = (PS', CL', \prec', \alpha')$ into another plan P'' by replacing plan step $ps \in PS'$ with $\alpha'(ps) = n_c$ if and only if:*

- *there is a plan $\widetilde{P} = (\widetilde{PS}, \widetilde{CL}, \widetilde{\prec}, \widetilde{\alpha})$ that is isomorphic to P, such that $\widetilde{PS} \cap PS' = \emptyset$,*
- *for each causal link $(ps', v, ps) \in in_{P'}(ps)$ there is a plan step $\widetilde{ps}_{(ps',v,ps)} \in \widetilde{PS}$, such that:*
  - *$v \in prec^+(\widetilde{\alpha}(\widetilde{ps}_{(ps',v,ps)}))$ in case $v \in prec^+(\alpha'(ps))$, or*
  - *$v \in prec^-(\widetilde{\alpha}(\widetilde{ps}_{(ps',v,ps)}))$ in case $v \in prec^-(\alpha'(ps))$,*
- *for each causal link $(ps, v, ps') \in out_{P'}(ps)$ there is a plan step $\widetilde{ps}_{(ps,v,ps')} \in \widetilde{PS}$, such that:*
  - *$v \in eff^+(\widetilde{\alpha}(\widetilde{ps}_{(ps,v,ps')}))$ in case $v \in eff^+(\alpha'(ps))$, or*
  - *$v \in eff^-(\widetilde{\alpha}(\widetilde{ps}_{(ps,v,ps')}))$ in case $v \in eff^-(\alpha'(ps))$,*
- *$P'' = (PS'', CL'', \prec'', \alpha'')$ is given as follows:*

$$\begin{aligned}
PS'' &:= (PS' \setminus \{ps\}) \cup \widetilde{PS} \\
CL'' &:= (CL' \setminus (in_{P'}(ps) \cup out_{P'}(ps))) \cup \widetilde{CL} \\
&\qquad \cup \{(ps', v, \widetilde{ps}_{(ps',v,ps)}) \mid (ps', v, ps) \in in_{P'}(ps)\} \\
&\qquad \cup \{(\widetilde{ps}_{(ps,v,ps')}, v, ps') \mid (ps, v, ps') \in out_{P'}(ps)\} \\
\prec'' &:= (\prec_1 \cup \widetilde{\prec} \cup \prec_2 \cup \prec_3)^*, \textit{ with} \\
&\prec_1 := (\prec' \setminus \{(ps', ps'') \in \prec' \mid \{ps\} \cap \{ps', ps''\} \neq \emptyset\}) \\
&\prec_2 := \{(ps', ps'') \in PS' \times \widetilde{PS} \mid (ps', ps) \in \prec'\} \cup \\
&\qquad \{(ps', ps'') \in \widetilde{PS} \times PS' \mid (ps, ps'') \in \prec'\} \\
&\prec_3 := \{(ps', \widetilde{ps}_{(ps',v,ps)}) \mid (ps', v, \widetilde{ps}_{(ps',v,ps)}) \in CL'' \\
&\qquad\qquad \textit{and } \{\alpha(ps'), \alpha(\widetilde{ps}_{(ps',v,ps)})\} \subseteq N_p\} \cup \\
&\qquad \{(\widetilde{ps}_{(ps,v,ps')}, ps') \mid (\widetilde{ps}_{(ps,v,ps')}, v, ps') \in CL'' \\
&\qquad\qquad \textit{and } \{\alpha(\widetilde{ps}_{(ps,v,ps')}), \alpha(ps')\} \subseteq N_p\} \\
\alpha'' &:= (\alpha' \setminus \{(ps, n_c)\}) \cup \widetilde{\alpha}
\end{aligned}$$

As noted, we require that causal links involving the decomposed plan step ps (i.e., $in_{P'}(ps)$ and $out_{P'}(ps)$) are passed down upon decomposition (cf. CL''). For this, any compatible precondition that is not yet protected by a causal link inside the method's plan $\widetilde{P}$ can be used. The definition of the ordering constraints of the new plan P'', $\prec''$ comprises all ordering constraints of the original plan P' except the ones involving the decomposed plan step ps, $\prec_1$. It further contains all ordering constraints of the method's plan $\widetilde{P}$, $\widetilde{\prec}$, as well as those that are inherited from the orderings involving ps, $\prec_2$. All new causal links only involving primitive tasks are responsible for adding further orderings, $\prec_3$. Apart from the necessary extensions to handle causal links, our definition of decomposition is identical to the one from HTN planning [19, Def. 3].

In HTN planning, any solution to a planning problem (1) needs to be obtainable from the initial task network via the application of a sequence of decompositions and (2) needs to contain an executable sequence of its actions [19, Def. 5, 6]. We consider the second criterion as impractical: One is usually interested in executable action *sequences*, but finding one from a "solution" task network is still $\mathbf{NP}$-**hard** [34, Thm. 15], [16, Thm. 8]. Further, such a sequence itself is in general not regarded a solution (but just the task network in which this sequence occurs), which we regard contra-intuitive. Instead, we require that *all* linearizations are executable and that each executable linearization is considered a solution as well. To support this stronger notion of solutions, we also allow the insertion of causal links and ordering constraints.

Definition 4 (Causal Link Insertion)**.** *Let $P = (PS, CL, \prec, \alpha)$ and $P' = (PS, CL', \prec', \alpha)$ be plans. P' can be obtained from P by insertion of a causal link $(ps, v, ps') \notin CL$ with $ps, ps' \in PS$ if and only if:*

- *$v \in eff^+(\alpha(ps))$ and $v \in prec^+(\alpha(ps'))$ or $v \in eff^-(\alpha(ps))$ and $v \in prec^-(\alpha(ps'))$,*
- *$CL' = CL \cup \{(ps, v, ps')\}$, and*
- *$\prec' = (\prec \cup \{(ps, ps') \mid \{\alpha(ps), \alpha(ps')\} \subseteq N_p\})^*$*

Definition 5 (Ordering Insertion)**.** *Let $P = (PS, CL, \prec, \alpha)$ and $P' = (PS, CL, \prec', \alpha)$ be plans. P' can be obtained from P by insertion of an ordering constraint $(ps, ps') \notin \prec$, $ps, ps' \in PS$ if and only if $\prec' = (\prec \cup \{(ps, ps')\})^*$.*

Definition 6 (Solution)**.** *A plan $P = (PS, CL, \prec, \alpha)$ is a solution to a planning problem π, if and only if:*

1. *P is a refinement of P^i. That is, there is a sequence of decompositions (cf. Def. 3), causal link insertions (cf. Def. 4), and ordering constraint insertions (cf. Def. 5) transforming P^i into P,*
2. *P is primitive, i.e., $\{\alpha(ps) \mid ps \in PS\} \subseteq N_p$, and*
3. *P is executable in the standard POCL sense:*
   - *There are no* unprotected preconditions. *A precondition $v \in prec^+(\alpha(ps))$ (resp. $v \in prec^-(\alpha(ps))$) of a plan step $ps \in PS$ is called unprotected, if and only if there is no plan step $ps' \in PS$ with a causal link (ps', v, ps) for $v \in eff^+(\alpha(ps))$ (resp. $v \in eff^-(\alpha(ps))$).*
   - *There are no* causal threats. *A plan contains a causal threat if and only if there is a causal link $(ps, v, ps') \in CL$ with $v \in prec^+(\alpha(ps'))$ (resp. $v \in prec^-(\alpha(ps'))$) and a plan step $ps'' \in PS$ with $v \in eff^-(\alpha(ps''))$ (resp. $v \in eff^+(\alpha(ps''))$), such that neither $(ps'', ps) \in \prec$ nor $(ps', ps'') \in \prec$ holds.*

The first solution criterion corresponds to the standard HTN criterion that requires every solution to be in the refinement space of the initial plan. The second solution criterion demands the respective plan to be primitive, as only primitive plans are typically regarded executable. The third criterion requires executability as it is done in POCL planning; these criteria ensure that every linearization of the plan steps corresponds to a sequence of tasks that is executable in the initial state and generates a state satisfying the goal description.

3 Legality Criteria – A Survey and Discussion

Provided the planning domain allows to specify preconditions and effects for compound tasks, there should be a clearly-defined criterion stating which decomposition methods are allowed to be specified for such compound tasks [17, 14]. When considering a compound task as an abstraction of a certain plan, it does not seem to make much sense to specify a precondition or effect of that task if it does not occur anywhere in the plan. So, if the domain modeler decides to specify preconditions and effects for a compound task, he or she has a certain idea on how the plans of the respective decomposition methods should look like. Thus, several researchers have formalized possible relations between a compound task's preconditions and effects and its methods' plans. We call these criteria *legality criteria* and plans that respect them *implementations* of their compound task. For each criterion, we give a small example illustrating it. Further, we want to note that the example depicted in Fig. 1 satisfies all of the legality criteria discussed in this section[¶].

The first and weakest criterion that we investigate is closely related to the criteria that ensure that a compound task can be decomposed (cf. Def. 3). We restrict to models where the plan of a compound task's method makes use of the task's preconditions and effects.

Definition 7 (Downward Compatible)**.** *A method $(n_c, P) \in M$ with $P = (PS, CL, \prec, \alpha)$ is called* downward compatible *if and only if:*

- *for each $v \in prec^+(n_c)$ (resp. $v \in prec^-(n_c)$) there is a plan step $ps \in PS$ with an unprotected precondition $v \in prec^+(\alpha(ps))$ (resp. $v \in prec^-(\alpha(ps))$).*
- *for each $v \in eff^+(n_c)$ (resp. $v \in eff^-(n_c)$) there is a plan step $ps \in PS$ with the same effect $v \in eff^+(\alpha(ps))$ (resp. $v \in eff^-(\alpha(ps))$).*

Downward compatible methods (n_c, P) are always applicable to a plan as long as it contains n_c – a property shared with standard HTN planning (without method preconditions). If the method was not downward compatible, causal links involving n_c influence its applicability, which also causes unintended "strange" behavior during search: Let us assume a plan P' contains a plan step ps with the name n_c that has an unprotected effect $v \in V$. Now, assume that (n_c, P) does violate the legality criterion. Whether this method can be used to generate a successor plan now depends on the planner's choice whether it first decomposes ps (this would work since all causal links can be correctly passed down to sub tasks in P') or first adds a causal link from $ps \in PS$ protecting v (then, ps can not be decomposed because the newly inserted link cannot be passed down, which violates the decomposition criteria given in Def. 3).

While this criterion clearly ensures that the "most obvious" modeling errors are prevented, it is not yet clear whether the user's intent about the relationship between the compound task and its methods' plans is actually satisfied. This is due to the fact that there are various possibilities what the preconditions and effects of the compound task are meant to entail. For example, if a primitive action n_p is within a plan, we know that – assuming that plan is executable – there are also states s and s', such that s satisfies n_p's precondition and s' satisfies n_p's effects. For compound tasks, this is not necessarily the case. In particular for the downward compatibility, it is clear that the compound task's effects do not need to be true in one single state, but its state variables might only hold in different states.

Several more restrictive criteria have been proposed in the literature. They can be categorized into two classes: one *enforces* that compound tasks have non-empty preconditions/effects under certain circumstances, and one where the specification thereof is *optional*.

We are only aware of one legality criterion that falls into the first class: It was proposed by Russell and Norvig for their fusion of HTN with POCL planning [37]. For each method (n_c, P) and every effect of n_c, they require that *"it [is] asserted by at least one step of P and is not denied by some other, later step"*. Further, they require that *"every precondition of the steps in P must be achieved by a step in P or be one of the preconditions of n_c"*. This implies that every open precondition in P needs to be a precondition of the compound task. This, in turn, has further consequences: Consider the case where the task n_c has two decomposition methods (n_c, P) and (n_c, P'). According to this criterion, n_c must use all open preconditions of both P and P'. As a consequence, both the downward compatibility and hence the monotonic property [27] may be violated. As argued in Sect. 2 (motivation for Def. 3), in this paper, we do not allow that commitments on an abstract level may be removed upon decomposition. We are thus not investigating this criterion in more detail.

The next criterion that we discuss was proposed by Biundo and Schattenberg [12]. As their planning formalism shows substantial differences to the one in this paper, we do not capture every detail, but only the main ideas. Their formalism is based upon a many-sorted first-order logic that features abstract state variables and so-called decomposition axioms defining them. They enable abstract tasks to make use of abstract state variables thereby fusing action abstraction with state abstraction. Further, their definition of methods only featured totally ordered plans: there, a method contains a set of task sequences, each of which is an implementation of the compound task. So, each method containing n sequences is a compact representation of n methods with totally ordered plans. Allowing for methods with partially ordered plans strictly increases expressivity (with respect to the plan existence problem [16, 2] and the generated solutions [21]), so legality should also be defined for such methods. We hence adapt their definition to partially ordered plans.

They require that if a method's task sequence is executable in a state satisfying the compound task's precondition, then it generates a state that satisfies the compound task's effects. They further require the compound task's precondition to be an abstraction of the task sequence's first task's precondition. Our generalization to partial orders states that this property must hold for every linearization that is induced by the given plan. However, we further demand that there also needs to be a state in which all linearizations are executable. As discussed earlier, causal links involving compound tasks do not (directly) induce ordering constraints (cf. Def. 1). However, since we consider a plan legal if all its linearizations are legal (which in turn need to respect the causal links), we here interpret causal links as additional ordering constraints. We thus define $\prec^+ = (\prec \cup \{(ps_i, ps_j) \mid (ps_i, v, ps_j) \in CL\})^*$ and only consider linearizations with respect to $\prec^+$. Then, a plan step linearization is said to *respect* a set of causal links CL if none of the protected conditions is violated by the induced state sequence.

Definition 8 (along Biundo and Schattenberg [12])**.** *A method $(n_c, P) \in M$, $P = (PS, CL, \prec, \alpha)$, is called* legal *if and only if:*

[¶] For the sake of readability, the example is given with variables, whereas our formalization assumes a ground (i.e., propositional) representation.

- *(n_c, P) is downward compatible,*
- *let $N_{\mathit{first}} := \{\alpha(ps) \mid ps \in PS$ and there is no $ps' \in PS$ with $(ps', ps) \in \prec^+\}$. Then, $prec^+(n_c) \subseteq \bigcap_{n \in N_{\mathit{first}}} prec^+(n)$ and $prec^-(n_c) \subseteq \bigcap_{n \in N_{\mathit{first}}} prec^-(n)$,*
- *for all states $s \in 2^V$ holds: if (a) $s \supseteq prec^+(n_c)$ and $s \cap prec^-(n_c) = \emptyset$, (b) every task sequence $\bar{t}$ corresponding to a plan step linearization $ps_1, \dots, ps_{|PS|}$ with respect to $\prec^+$ is executable in s, and (c) respects CL, then (d) $\bar{t}$ generates a state s', such that $\mathit{eff}^+(n_c) \subseteq s'$ and $\mathit{eff}^-(n_c) \cap s' = \emptyset$. Further, there needs to be a state $s \in 2^V$, such that (a) to (d) hold.*

Concerning the intention behind the compound task's preconditions and effects, this criterion is already much stronger than the simple downward compatibility criterion. As opposed to that criterion, we here get the property that in any solution plan that is obtained from decomposing a compound task n_c, there is a *single* state in which n_c's preconditions hold. This trivially follows from the fact that the preconditions of n_c are abstractions of any first task and any compound task needs to be decomposed into a primitive plan. However, the criteria do not imply that there is a single state in which the compound task's effects hold – despite the restrictions imposed on the relationship between n_c's effects and its methods' plans. The reason for this is that the criterion does not take into account additional effects of the compound tasks that are in the method's plan, other than those explicitly specified in their preconditions and effects. This issue is addressed by Marthi et al.'s *possible* effects [30]. They are, however, restricting to totally ordered methods.

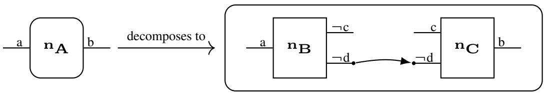


Figure 2. Illustration of a method that is supposed to implement the compound task n_A. The tasks n_B and n_C are primitive.

Concerning executability, Def. 8 requires that there is at least one state in which the respective plan's linearizations are executable, which might be considered too restrictive. The example given in Fig. 2 is not legal with respect to Def. 8, since – due to the variable c – there cannot be a state in which n_A's method's plan is executable. Since other tasks could be ordered in between n_B and n_C to support n_C's precondition c, one could also relax this criterion. This is done by the next legality criterion (for which the example is legal), as it allows that open preconditions of a method's plan may be supported by tasks at an arbitrary "position" within a plan (i.e., it does not require that a single state enables the execution of the plan's tasks).

The criterion was proposed by Yang [40, p. 14] and consists of three sub criteria. The first two imply downward compatibility, but require more. Criterion 1 demands, similar to Def. 8, that none of the tasks in the plan invalidates the compound task's effects. Criterion 2 requires that every precondition variable of the compound task has to occur in the plan in such a way that none of its sub tasks can be used to establish it. Criterion 3 is closely related to the concept of causal threats. Each precondition within the plan might not be violated by a converse effect of another task. Note that this is neither equivalent to stating that the respective plan must be free of causal threats (as the criterion must also hold in the absence of any causal links) nor that the plan is executable in any way (as none of the criteria ensures preconditions to be supported).

Definition 9 (Yang [40]). *A method $(n_c, P) \in M$ with $P = (PS, CL, \prec, \alpha)$ is called* legal *if and only if:*

1. *for all $v \in \mathit{eff}^+(n_c)$ (resp. $v \in \mathit{eff}^-(n_c)$) there exists a $ps \in PS$ with $v \in \mathit{eff}^+(\alpha(ps))$ (resp. $v \in \mathit{eff}^-(\alpha(ps))$), such that for all $ps' \in PS$, $ps' \neq ps$ holds: if $v \in \mathit{eff}^-(\alpha(ps'))$ (resp. $v \in \mathit{eff}^+(\alpha(ps'))$), then $(ps', ps) \in \prec$.*
2. *for all $v \in prec^+(n_c)$ (resp. $v \in prec^-(n_c)$) there exists a $ps \in PS$ with $v \in prec^+(\alpha(ps))$ (resp. $v \in prec^-(\alpha(ps))$), such that for all $ps' \in PS$, $ps' \neq ps$ holds: if $v \in \mathit{eff}^+(\alpha(ps'))$ (resp. $v \in \mathit{eff}^-(\alpha(ps'))$), then $(ps, ps') \in \prec$.*
3. *for all $ps \in PS$, for all $v \in prec^+(\alpha(ps))$ (resp. $v \in prec^-(\alpha(ps))$), and for all $ps' \in PS$ with $ps' \neq ps$ and $(ps, ps') \notin \prec$ it holds: if $v \in \mathit{eff}^-(\alpha(ps'))$ (resp. $v \in \mathit{eff}^+(\alpha(ps'))$), then there exists a $ps'' \in PS$, such that $\{(ps', ps''), (ps'', ps)\} \subseteq \prec$ and $v \in \mathit{eff}^+(\alpha(ps''))$ (resp. $v \in \mathit{eff}^-(\alpha(ps''))$).*

The next legality criterion is by Young et al. [41]. They argue that Yang's criterion of threat-free plans was too strong. Instead, their only requirement is that any of the compound task's preconditions "contributes" to at least one of its effects (and vice versa), which they ensure by requiring the existence of a chain of causal links within the plans connecting them with each other [41, p. 191]. Thus, our example illustrated in Fig. 2 also satisfies this criterion, but it would not do so if both the variable d and the respective causal link were not present (while assuming that n_C is still ordered behind n_B).

They model their criterion by including artificial start and end actions, which use the compound task's precondition and effect as effect and precondition, respectively, and require the causal link chains between them. Upon decomposition, those actions disappear, but the causal links involving them are reused to be linked to the plan steps that share causal links with the compound task. Those actions thus do not imply that there are states, in which the compound task's preconditions and effects hold. In the following definition, we therefor did not include the artificial start and end tasks.

Definition 10 (Young et al. [41]). *A method $(n_c, P) \in M$ with $P = (PS, CL, \prec, \alpha)$ is called* legal *if and only if:*

- *(n_c, P) is downward compatible and*
- *for each $v \in \mathit{eff}^+(n_c)$ (resp. $v \in \mathit{eff}^-(n_c)$), there is a sequence of plan steps $ps_1, \dots, ps_k$ with $ps_i \in PS$ for $1 \leq i \leq k$, $v \in \mathit{eff}^+(\alpha(ps_k))$ (resp. $v \in \mathit{eff}^-(\alpha(ps_k))$), $v' \in (prec^+(\alpha(ps_1)) \cup prec^-(\alpha(ps_1))) \cap (prec^+(n_c) \cup prec^-(n_c))$, and a chain of causal links connecting them.*
- *for each $v \in prec^+(n_c)$ (resp. $v \in prec^-(n_c)$), there is a sequence of plan steps $ps_1, \dots, ps_{k'}$ with $ps_i \in PS$ for $1 \leq i \leq k'$, $v \in prec^+(\alpha(ps_1))$ (resp. $v \in prec^-(\alpha(ps_1))$), $v' \in (\mathit{eff}^+(\alpha(ps_{k'})) \cup \mathit{eff}^-(\alpha(ps_{k'}))) \cap (\mathit{eff}^+(n_c) \cup \mathit{eff}^-(n_c))$, and a chain of causal links connecting them.*

Due to space restrictions, we cannot include all the work related to formalisms that allow to specify preconditions and effects for compound tasks. Concerning the surveyed legality criteria, we restricted the presentation to approaches which *explicitly* mention the demanded criteria. We further want to mention the work by Castillo et al. [14], which also fuses HTN planning with POCL planning. Since they infer the methods automatically, their plans also fulfill certain criteria, which seem to be closely related to Def. 8. Further attention deserves the *angelic semantics* by Marthi et al. [30]. Their conditions ("high-level action descriptions") take all states into account that are generated by any primitive plan that is reachable by decomposing the respective compound task. In contrast, the legality criteria surveyed before relate a compound task's preconditions and effects directly to its methods' plans (in particular to the preconditions and effects of its tasks).

4 Complexity Results

In this section we investigate the complexity of the plan verification and the plan existence problem for the hybrid planning formalism.

For some of our hardness results, we reduce a certain set of HTN planning problems to hybrid problems. Since all required results are proved or reproduced in the HTN planning framework by Geier and Bercher [19] (or based upon it), we first state a proposition that every HTN planning problem can be expressed as a hybrid planning problem with the same set of solutions and without violating any of the legality criteria for decomposition methods. Concerning notation, note that a *task network* $(T, \prec, \alpha)$ [19, Def. 1] in HTN planning is a special case of plans (cf. Def. 1), as task networks do not contain causal links. What we call plan steps is referred to as *tasks* T in HTN planning. We use both notations, depending on the context.

Theorem 1. *Let $\mathcal{P}$ be an HTN planning problem according to Def. 2 by Geier and Bercher [19]. Then, $\mathcal{P}$ can be transformed into a hybrid planning problem π according to Def. 2 that satisfies the legality criteria in Defs. 7, 8, 9, and 10, such that:*

1. ***Refinement Correspondence.** Let tn be a (not necessarily primitive) task network that can be obtained via decomposition in $\mathcal{P}$. Then, the plan P that is isomorphic to tn can be obtained via decomposition in π. Conversely, let P be a primitive plan that can be obtained via decomposition in π. Then, the task network tn that is isomorphic to P can be obtained via decomposition in $\mathcal{P}$.*
2. ***Solution Correspondence.** Let tn be a solution task network for $\mathcal{P}$. Then, for each executable task sequence $t_1, \ldots, t_n$ thereof there is a solution plan P to π containing exactly this task sequence. Conversely, let P be a solution plan for π. Then, for all linearizations $\overline{ps}$ of the plan steps of P there is a task network tn that is a solution for $\mathcal{P}$, such that $\overline{ps}$ is an executable linearization of the tasks in tn.*

Proof. Let $\mathcal{P} = (L, C, O, M, c_I, s_I)$ with L being a finite set of proposition symbols and C and O finite sets of compound and primitive task name symbols, respectively. Each primitive task name $o \in O$ has a unique operator $(prec(o), add(o), del(o)) \in (2^L)^3$ corresponding to an action without negative preconditions. M is a finite set of methods mapping compound tasks to task networks. $c_I \in C$ is the initial compound task name and $s_I \in 2^L$ the initial state.

Transformation. We transform the HTN planning problem $\mathcal{P}$ into a hybrid planning problem $\pi = (V, N_c, N_p, \delta, M', P^i)$. We define $V := L$. The initial compound task c_I and the initial state s_I of $\mathcal{P}$ are encoded in π by the initial plan $P^i = (PS^i, CL^i, \prec^i, \alpha^i)$. That is, $PS^i = \{ps_{init}, ps, ps_{goal}\}$, $CL^i = \emptyset$, $\prec^i = \{(ps_{init}, ps), (ps, ps_{goal})\}^*$, and $\alpha^i = \{(ps_{init}, init), (ps, c_I), (ps_{goal}, goal)\}$. The actions of $init$ and $goal$ are given by $\delta(init) = (\emptyset, \emptyset, s_I, V \setminus s_I)$ and $\delta(goal) = (\emptyset, \emptyset, \emptyset, \emptyset)$. We are now defining the tasks and methods of π in such a way that all methods in M' satisfy all legality criteria. We construct a model in which no compound task has preconditions or effects and all methods contain only compound tasks or at most one task. We do this by introducing an additional compound task o_{clone} for every primitive task $o \in O$ and replace all primitive tasks in every decomposition method's plan by the respective compound task. To ensure that this does not change the set of solutions, each of these compound tasks o_{clone} has exactly one decomposition method with a plan containing exactly o: Let $N_c := C \cup \{o_{clone} \mid o \in O\}$ and for all $n_c \in N_c$, let $\delta(n_c) := (\emptyset, \emptyset, \emptyset, \emptyset)$. For the primitive tasks, let $N_p := O \cup \{init, goal\}$ and for all $o \in O$, let $\delta(o) = (prec(o), \emptyset, add(o), del(o))$. The methods M' are given by

$$\begin{aligned} M' := \{(c, (T, \emptyset, \prec, \alpha')) \mid (c, (T, \prec, \alpha)) \in M, \text{ with} \\ \alpha' := \{(t, c) \mid (t, c) \in \alpha, c \in C\} \cup \\ \{(t, o_{clone}) \mid (t, o) \in \alpha, o \in O\}\} \\ \cup \{(o_{clone}, (\{ps\}, \emptyset, \emptyset, \{(ps, o)\})) \mid o \in O\} \end{aligned}$$

Legality. The downward compatibility (Def. 7) and legality criterion that requires chains of causal links (Def. 10) trivially hold for all methods, since none of the compound tasks have preconditions or effects (so, the methods' plans are not further restricted). The other legality criteria (Defs. 8 and 9) require further restrictions on the plans even if the (parent) compound task does not have preconditions or effects. It is easy to see that all these restrictions hold, since – by construction – plans only contain compound tasks or at most one task. In the first case, there are no preconditions and effects, hence all criteria hold. In the second, the preconditions or effects of the single task do not violate any of the criteria as well.

Refinement Correspondence. Let $tn_1, \ldots, tn_k$ be a sequence of task networks, such that $tn_1 = (\{t\}, \emptyset, \{(t, c_I)\})$, $tn_k = tn$, and any task network tn_j can be obtained from tn_{j-1}, $1 < j \leq k$, via decomposition. Let the corresponding sequence of decomposed task names be $c_1, \ldots, c_{k-1}$. Since no compound task $c \in C$ was removed from any of the decomposition methods' task networks, and since none of them contains causal links, there is also a sequence of plans $P_1, \ldots P_k$, such that $P_1 = P^i$ and the plan P_j results from decomposing c_{j-1} in P_{j-1}, $1 < j \leq k$. The resulting plan P_k is isomorphic to the task network tn_k except that P_k contains a compound task $o_{clone} \in N_c \setminus C$ for any primitive task $o \in O$ in tn_k. Thus, using the methods for those $o_{clone} \in N_c \setminus C$, there is a sequence of decompositions that transform P_k into P'_k with P'_k being isomorphic to tn_k. For the other direction, let P be a primitive plan that can be obtained via decomposition in π. Because the decomposition of a compound task $o_{clone} \in N_c \setminus C$ does not introduce further compound tasks, we can assume that first tasks $c_1, \ldots, c_{k-1}$, $c_i \in C$, $1 \leq i < k$ are decomposed leading to a plan P_k and then only tasks in $o_{clone} \in N_c \setminus C$ leading to a primitive plan P'_k. When decomposing $c_1, \ldots, c_{k-1}$ in $\mathcal{P}$, we obtain a task network tn_k that is isomorphic to P'_k.

Solution Correspondence. Let tn be a solution to $\mathcal{P}$. It is thus reachable via decomposition in $\mathcal{P}$. Due to the refinement correspondence, P being isomorphic to tn can be obtained via decomposition in π. Thus, for any executable linearization of the tasks of tn, P can be turned into a totally ordered solution plan P' containing exactly that sequence via ordering and causal link insertions. For the other direction, let P be a solution for π. Without loss of generality we can assume that P can be generated by first decomposing, then inserting causal links, then ordering constraints. Let P' be the last primitive plan before any ordering or causal link insertion. Due to the refinement correspondence, tn being isomorphic to P' is reachable via decomposition in $\mathcal{P}$. Then, tn contains all linearizations of the plan steps of P' (in particular the executable ones). □

Plan Verification. We now investigate how hard it is to verify whether a given plan is a solution to a hybrid planning problem. While in classical, non-hierarchical planning, this question can be answered in linear time w.r.t. the size of the input plan [16, Thm. 8], the corresponding problem is much harder in the HTN setting. Behnke et al. [9] proved that the HTN plan verification problem is $\mathbf{NP}$-$\mathbf{complete}$ even under several restrictions.

We first investigate the special case where there is no hierarchy. In HTN planning, this means to decide whether a primitive plan has an executable linearization, which is already $\mathbf{NP}$-$\mathbf{complete}$ [34, Thm. 15], [16, Thm. 8]. In hybrid planning, *all* linearizations need to be executable. The respective problem is hence equivalent to verifying whether a POCL plan is a solution to a POCL planning problem, which is commonly known to be tractable.

Theorem 2. *Let P be a plan and $\pi = (V, N_c, N_p, \delta, M, P^i)$ a hybrid planning problem without hierarchy, i.e., $N_c = M = \emptyset$. Deciding whether P is a solution to π is in* $\mathbf{P}$.

Proof. Checking that every precondition is supported by exactly one causal link can be done in linear time w.r.t. the number of all preconditions and causal links. Checking the absence of causal threats can be done in quadratic time. Let $P = (PS, CL, \prec, \alpha)$. For each causal link $(ps, v, ps') \in CL$ iterate over all plan steps $ps'' \in PS$, $ps'' \notin \{ps, ps'\}$. If none of these ps'' has an effect conflicting with v and can be ordered between ps and ps' without violating $\prec$, continue to the next causal link, otherwise fail. □

Please note that the reason why this verification problem is easier than in HTN planning cannot be attributed to the fact that compound tasks show preconditions and effects, but to the fact that in hybrid (and POCL) planning, all linearizations need to be executable, whereas HTN planning only requires that there exists one.

Next we consider the general case, in which there are no restrictions on the hierarchy. We start by showing $\mathbf{NP}$ membership.

Lemma 1. *Let P be a plan and π a hybrid planning problem. Independently of the demanded legality criteria (Def. 7 to 10), it is in* $\mathbf{NP}$ *to decide whether P is a solution to π.*

Proof sketch: For HTN planning, we showed that the corresponding verification problem is in $\mathbf{NP}$ [9, Thm. 1]. We show that the two main proof steps (not emphasizing some special cases due to lack of space) remain applicable despite the extension of the formalism to hybrid planning: First, we show that any plan that can be obtained via decomposition can be obtained by a polynomial number of methods. Second, we give a guess-and-verify algorithm that runs in $\mathbf{NP}$.

The first main step consists of five sub steps. First, we construct a so-called ε-extended planning problem π' from π that has the same set of solutions as π, but allows for shorter decomposition sequences. We call methods that contain an empty plan ε-methods. Now, for any compound task name that can be transformed into an empty plan by an arbitrary number of decomposition methods (due to ε-methods already present in M), we introduce an additional ε-method in M' of π'. For HTN planning, this can be done in $\mathbf{P}$ [9, after Def. 1]. Since causal links are not allowed to disappear upon decomposition, the same result applies to hybrid planning. Second, given a sequence $\overline{m}_1$ of methods in M' leading from the initial plan to a plan P', the methods are reordered as follows: all ε-methods immediately follow the method that inserted the plan step they erase into the plan, resulting in the sequence $\overline{m}_2$. Note that reordering these decomposition methods is possible, since all legality criteria satisfy Def. 7, downward compatibility (cf. footnote on page 4). Third, if for any non-ε-method in $\overline{m}_2$, all plan steps of its plan are thereafter erased, all those methods are replaced by one singe ε-method resulting in a shorter sequence $\overline{m}_3$. Let $\overline{P} = P_1, \ldots, P_n$ with $P_1 = P^i$ and $P_n = P'$ be the corresponding sequence of plans. Its subsequence $\overline{P}'$ that consists of the plans to which non-ε-methods are applied and P' forms a sequence with non-decreasing plan step size. Due to plateaus (which can be caused, e. g., by so-called unit-methods, which decompose a compound task into a single other task), $\overline{P}'$ can still be arbitrary long. Step four is a preparation to obtain a bound on their lengths: We reorder methods between the plateaus such that in every plateau only methods remain that decompose the plan step that is decomposed last in the respective plateau (as this step ends the plateau) resulting into $\overline{m}_4$. As argued before, reordering is also possible in the hybrid planning setting. In step five we can now shorten the new sequence of plans corresponding to $\overline{m}_4$. Every plateau in the new plan sequence can now be limited to at most $|N_c|$ plans, as otherwise cycles must occur. Because there are at most k plateaus (k being $|PS|$ of P'), we can state that the final sequence of methods $\overline{m}_5$ has at most $2k(|N_c|+1)$ non-ε-methods[‖] and thus $2k(|N_c|+1)\Delta$ methods in total, Δ being the maximal number of plan steps of the plans in the methods and P^i. We thereby conclude that if a plan P' can be obtained via decomposition in π, it can be obtained by a polynomial number of methods in π'.

For the second main step, we guess a polynomially bounded sequence of methods in π' and calculate the resulting plan P'. We then guess additional ordering constraints (bounded by k^2) and causal links (bounded by $k|V|$), insert them into P' resulting in P'', guess a bijection between the plan steps in P'' and P and verify isomorphism. We then verify executability of P in $\mathbf{P}$ (Thm. 2). □

We now show that the plan verification problem is also $\mathbf{NP}$-$\mathbf{hard}$.

Theorem 3. *Let P be a plan and π a hybrid planning problem. Independently of the demanded legality criteria (Def. 7 to 10), it is* $\mathbf{NP}$-$\mathbf{complete}$ *to decide whether P is a solution to π.*

Proof. Membership is stated in Lem. 1. For hardness, we use a corollary of HTN plan verification [9, Cor. 5]. According to this, it is $\mathbf{NP}$-$\mathbf{complete}$ to verify, given a sequence of tasks $\bar{t}$ and a totally unordered precondition- and effect-free HTN problem $\mathcal{P}$, whether there is a solution task network tn, such that $\bar{t}$ is an (executable) linearization of tn's tasks. (Note that the complexity of the problem does not stem from finding an *executable* linearization, as there are no preconditions and effects, but from finding the right decompositions leading to the desired plan. The original proof reduces vertex cover to HTN plan verification.)

Let π be a hybrid planning problem that is constructed from $\mathcal{P}$ with the properties stated in Thm. 1. Further, let P be a totally ordered plan containing $\bar{t}$ as plan step sequence. If there is a solution tn of $\mathcal{P}$, such that $\bar{t}$ is an executable linearization of tn, then we can conclude that P is a solution to π (Thm. 1). Conversely, if P is a solution to π, then there exists a solution tn to $\mathcal{P}$, such that P's plan step linearization is an executable linearization of tn (Thm. 1). □

Plan Existence. In general, HTN planning is $\mathbf{undecidable}$ [16, 19]. We now show that this also holds for hybrid planning.

Theorem 4. *Hybrid planning is* $\mathbf{undecidable}$. *That is, it is undecidable to determine whether a hybrid planning problem has a solution – no matter, which of the legality criteria of Def. 7 to 10 hold.*

Proof. Since we can encode any HTN planning problem into a solution-conserving hybrid planning problem that satisfies all legality criteria (Thm. 1), we can reduce the undecidable plan existence problem for HTN planning [19, Thm. 1] to hybrid planning. □

[‖] We previously stated a bound of only $|k|(|N_c|+1)$ [9, Lem. 1], as we handled a special case wrong (details omitted due to space restrictions).

From this theorem we can conclude that hybrid planning is as expressive as HTN planning, since it allows to encode undecidable problems. However, HTN planning is also known to be semi-decidable (or recursively enumerable, $\mathbf{RE}$) [16, Thm. 1], which implies that for any HTN planning problem, a solution can be eventually found if one exists (while the undecidability prevents one from proving – in general – that there is no solution in case there actually is none). We now show that this is also true for hybrid planning.

Theorem 5. *Hybrid planning is* $\mathbf{semi\text{-}decidable}$*. That is, the set of all hybrid planning problems that possess a solution is in* $\mathbf{RE}$*.*

Proof. We give a partial recursive function f that, given a hybrid planning problem π, returns *true* if π has a solution and that may not halt, otherwise. We define f as the algorithm that enumerates all plans and verifies whether they solve π. It may run infinitely long, but as soon as it finds a solution, f returns *true*. Although there are infinitely many plans, enumeration is possible by starting with all plans of length two (the only tasks of which are the artificial *init* and *goal* actions) and then successively incrementing plan length. For each plan, verify in $\mathbf{NP}$ whether it is a solution (Thm. 3). □

As a further corollary from Thm. 1, it also follows that many sub classes of hybrid planning are as hard as the respective problem classes in HTN planning. Such restrictions include syntactical ones (such as totally ordered task networks [2] or delete-relaxed actions [5]) and structural restrictions on the hierarchy (such as tail-recursive or acyclic problems [2]). To formally prove this, we would have to show that the respective restrictions still hold in the hybrid planning problem after the translation process done in the proof of Thm. 1.

5 Discussion

As a corollary from the last section's results, we can observe that for the studied legality criteria, allowing preconditions and effects for compound tasks does neither increase nor decrease the expressivity of the formalism with regard to the plan existence problem in the general case. We want to emphasize that this is caused by the fact that none of the studied criteria (Def. 7 to 10) *enforces* to specify preconditions or effects for compound tasks (such as the one by Russell and Norvig [37]). Legality criteria that enforce to specify such preconditions or effects might influence the respective results and therefore also reduce expressivity, as they might prevent to specify computationally hard problems. Having *the option* to model such preconditions and effects still serves several practically relevant purposes, however. We shortly discuss some of them in this section and give pointers to the literature for further details.

As argued by Fox [17, p. 196], *"one of the strongest motivations for using some form of abstraction in planning is the observation that people use it to great effect in their problem-solving"*, which is also backed up by psychological studies [13]. Consequently, people should already be supported during the process of constructing (hierarchical) planning domains. Modeling support has attracted increased interest during the last years, which is one of the reasons that lead to the establishment of the *International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*[**]. Nevertheless, there is still only very little research to automatically support the modeling process, which is particularly true for hierarchical models. The tool by McCluskey and Kitchin [32] as well as GIPO [39] for hierarchical models expressed in the modeling language OCL$_h$ checks certain properties and reports violations. In analogy to the respective properties that these tools verify, the legality criteria allow to automatically verify the relationship between compound tasks and their methods' plans. As Fox points out, *"abstract plans have intentional meaning"* – and so do compound tasks. Thus, adhering some desired legality criterion is a possible way to automatically verify whether the model complies with the user's intent.

Apart from providing modeling assistance, another main purpose of being able to specify preconditions and effects for compound tasks is to exploit them during search. Reasoning about these preconditions and effects may result in a smaller search space, as irresolvable flaws, such as open preconditions, can be detected earlier, i.e., before the respective compound task is decomposed. This further allows to generate "solution" plans on different levels of abstraction. Such plans look like ordinary (primitive) solutions with the difference that some tasks are still compound. When the model fulfills further prerequisites, it is guaranteed that such abstract solutions can be refined into a primitive one [40, 30]. Then, the model is said to fulfill the downward refinement property [6].

Preconditions and effects of compound tasks can also improve plan explanations. Plan explanations as developed by Seegebarth et al. [38] give a justification about the purpose of a primitive action questioned by the user. It is based upon a sequence of arguments, each being of the form (a) "action a is required as it supports a precondition variable of another action a' by a causal link" or (b) "action a is required as it was introduced via decomposition of a compound task c". Necessity of the other task a' (resp. c) is proved similarly. Preconditions and effects of compound tasks now allow to combine these two argument types [38]. Then, the causal chain argument (a) can be extended from just primitive actions to compound tasks, as they show preconditions and effects as well, which allows for much shorter and more abstract explanations.

6 Conclusion

To finally answer the question whether compound tasks with preconditions and effects are more than just names: We can state *no* in the sense that for the criteria we studied in more detail, we were able to show that in the general case, the formalism is equally expressive (with respect to the plan existence problem) than the HTN formalism, in which compound tasks are just names. For many sub classes, however, we only showed lower bounds – upper bounds still need to be proved. It might also be that other, more restrictive, legality criteria influence the hardness of the problem, in which case we also had to state *yes*. We can already answer the question with *yes* with regard to practical considerations, such as modeling assistance: The preconditions and effects, when combined with a desired legality criterion, can be exploited to provide assistance to ensure that the methods comply with the user's intent – or at least to rule out some of the modeling flaws.

ACKNOWLEDGEMENTS

We thank Thomas Geier for discussions that helped improving this paper as well as Kathi Krammer for proof reading an early draft. This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

We also thank Mario Schmautz for contributing to this revised version, most notably in Defs. 3 and 10.

[**] http://www.icaps-conference.org/index.php/Main/Competitions

REFERENCES

[1] Ron Alford, Gregor Behnke, Daniel Höller, Pascal Bercher, Susanne Biundo, and David Aha, 'Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems', in *Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 20–28. AAAI Press, (2016).

[2] Ron Alford, Pascal Bercher, and David Aha, 'Tight bounds for HTN planning', in *Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 7–15. AAAI Press, (2015).

[3] Ron Alford, Pascal Bercher, and David Aha, 'Tight bounds for HTN planning with task insertion', in *Proc. of the 25th Int. Joint Conf. on AI (IJCAI)*, pp. 1502–1508. AAAI Press, (2015).

[4] Ron Alford, Ugur Kuter, and Dana S. Nau, 'Translating HTNs to PDDL: A small amount of domain knowledge can go a long way', in *Proc. of the 21st Int. Joint Conf. on AI (IJCAI)*, pp. 1629–1634. AAAI Press, (2009).

[5] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana Nau, 'On the feasibility of planning graph style heuristics for HTN planning', in *Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 2–10. AAAI Press, (2014).

[6] Fahiem Bacchus and Qiang Yang, 'Downward refinement and the efficiency of hierarchical problem solving', *Artificial Intelligence*, **71**(1), 43 – 100, (1994).

[7] Patrick Bechon, Magali Barbier, Guillaume Infantes, Charles Lesire, and Vincent Vidal, 'HiPOP: Hierarchical partial-order planning', in *Proc. of the 7th Europ. Starting AI Researcher Symposium (STAIRS)*. IOS Press, (2014).

[8] Gregor Behnke, Daniel Höller, Pascal Bercher, and Susanne Biundo, 'Change the plan – how hard can that be?', in *Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 38–46. AAAI Press, (2016).

[9] Gregor Behnke, Daniel Höller, and Susanne Biundo, 'On the complexity of HTN plan verification and its implications for plan recognition', in *Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 25–33. AAAI Press, (2015).

[10] Pascal Bercher, Susanne Biundo, Thomas Geier, Thilo Hörnle, Florian Nothdurft, Felix Richter, and Bernd Schattenberg, 'Plan, repair, execute, explain - how planning helps to assemble your home theater', in *Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 386–394. AAAI Press, (2014).

[11] Susanne Biundo, Pascal Bercher, Thomas Geier, Felix Müller, and Bernd Schattenberg, 'Advanced user assistance based on AI planning', *Cognitive Systems Research*, **12**(3-4), 219–236, (2011). Special Issue on Complex Cognition.

[12] Susanne Biundo and Bernd Schattenberg, 'From abstract crisis to concrete relief – a preliminary report on combining state abstraction and HTN planning', in *Proc. of the 6th Europ. Conf. on Planning (ECP)*, pp. 157–168. AAAI Press, (2001).

[13] Richard Byrne, 'Planning meals: Problem solving on a real data-base', *Cognition*, **5**, 287–332, (1977).

[14] Luis A. Castillo, Juan Fernández-Olivares, and Antonio González, 'On the adequacy of hierarchical planning characteristics for real-world problem solving', in *Proc. of the 6th Europ. Conf. on Planning (ECP)*, pp. 169–180. AAAI Press, (2001).

[15] Filip Dvořk, Arthur Bit-Monnot, Flix Ingrand, and Malik Ghallab, 'A flexible ANML actor and planner in robotics', in *Proc. of the 2nd Workshop on Planning and Robotics (PlanRob)*, pp. 12–19, (2014).

[16] Kutluhan Erol, James A. Hendler, and Dana S. Nau, 'Complexity results for HTN planning', *Annals of Mathematics and Artificial Intelligence*, **18**(1), 69–93, (1996).

[17] Maria Fox, 'Natural hierarchical planning using operator decomposition', in *Proc. of the 4th Europ. Conf. on Planning (ECP)*, pp. 195–207. Springer, (1997).

[18] Maria Fox and Derek Long, 'Hierarchical planning using abstraction', *IEE Proc. – Control Theory Appl.*, **142**(3), 197–210, (1995).

[19] Thomas Geier and Pascal Bercher, 'On the decidability of HTN planning with task insertion', in *Proc. of the 22nd Int. Joint Conf. on AI (IJCAI)*, pp. 1955–1961. AAAI Press, (2011).

[20] Malik Ghallab, Dana S. Nau, and Paolo Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.

[21] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo, 'Language classification of hierarchical planning problems', in *Proc. of the 21st Europ. Conf. on AI (ECAI)*, pp. 447–452. IOS Press, (2014).

[22] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo, 'Assessing the expressivity of planning formalisms through the comparison to formal languages', in *Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 158–165. AAAI Press, (2016).

[23] Éric Jacopin, 'Game AI planning analytics: The case of three first-person shooters', in *Proc. of the 10th AI and Interactive Digital Entertainment Conf. (AIIDE)*, pp. 119–124. AAAI Press, (2014).

[24] Subbarao Kambhampati, 'Refinement planning as a unifying framework for plan synthesis', *AI Magazine*, **18**(2), 67–98, (1997).

[25] Subbarao Kambhampati and James A. Hendler, 'A validation-structure-based theory of plan modification and reuse', *Artificial Intelligence*, **55**, 193–258, (1992).

[26] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava, 'Hybrid planning for partially hierarchical domains', in *Proc. of the 15th Nat. Conf. on AI (AAAI)*, pp. 882–888. AAAI Press, (1998).

[27] Craig A. Knoblock, 'Automatically generating abstractions for planning', *Artificial Intelligence*, **68**, 243–302, (1994).

[28] Raphaël Lallement, Lavindra De Silva, and Rachid Alami, 'HATP: An HTN planner for robotics', in *Proc. of the 2nd Workshop on Planning and Robotics (PlanRob)*, pp. 20–27, (2014).

[29] Naiwen Lin, Ugur Kuter, and Evren Sirin, 'Web service composition with user preferences', in *Proc. of the 5th Europ. Semantic Web Conf. (ESWC)*, pp. 629–643. Springer, (2008).

[30] Bhaskara Marthi, Stuart J. Russell, and Jason Wolfe, 'Angelic semantics for high-level actions', in *Proc. of the 17nd Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 232–239. AAAI Press, (2007).

[31] David McAllester and David Rosenblitt, 'Systematic nonlinear planning', in *Proc. of the 9th Nat. Conf. on AI (AAAI)*, pp. 634–639. AAAI Press, (1991).

[32] Thomas Lee McCluskey and Diane E. Kitchin, 'A tool-supported approach to engineering HTN planning models', in *In Proc. of 10th IEEE Int. Conf. on Tools with AI (ICTAI)*, pp. 272–279. IEEE, (1998).

[33] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Dan Wu, Fusun Yaman, Héctor Muñoz-Avila, and J. William Murdock, 'Applications of SHOP and SHOP2', *Intelligent Systems, IEEE*, **20**, 34–41, (2005).

[34] Bernhard Nebel and Christer Bäckström, 'On the computational complexity of temporal projection, planning, and plan validation', *Artificial Intelligence*, **66**(1), 125–160, (1994).

[35] Santiago Ontañón and Michael Buro, 'Adversarial hierarchical-task network planning for complex real-time games', in *Proc. of the 24th Int. Conf. on AI (IJCAI)*, pp. 1652–1658. AAAI Press, (2015).

[36] J. Scott Penberthy and Daniel S. Weld, 'UCPOP: A sound, complete, partial order planner for ADL', in *Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pp. 103–114. Morgan Kaufmann, (1992).

[37] Stuart Russell and Peter Norvig, *Artificial Intelligence – A modern Approach*, chapter 12: Practical Planning, 367–376, Prentice-Hall, 1 edn., 1994.

[38] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo, 'Making hybrid plans more clear to human users – a formal approach for generating sound explanations', in *Proc. of the 22nd Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pp. 225–233. AAAI Press, (2012).

[39] Ron M. Simpson, Diane E. Kitchin, and Thomas Lee McCluskey, 'Planning domain definition using GIPO', *The Knowledge Engineering Review*, **22**, 117–134, (2007).

[40] Qiang Yang, 'Formalizing planning knowledge for hierarchical planning', *Computational Intelligence*, **6**(1), 12–24, (1990).

[41] Robert Michael Young, Martha E. Pollack, and Johanna D. Moore, 'Decomposition and causality in partial-order planning', in *Proc. of the 2nd Int. Conf. on AI Planning Systems (AIPS)*, pp. 188–193. AAAI Press, (1994).

The following pages show the publication:

P. Bercher, T. Geier, F. Richter, and S. Biundo. “On Delete Relaxation in Partial-Order Causal-Link Planning”. In: *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013).* IEEE Computer Society, 2013, pp. 674–681. DOI: 10.1109/ICTAI.2013.105

On Delete Relaxation in Partial-Order Causal-Link Planning

Pascal Bercher, Thomas Geier, Felix Richter, Susanne Biundo
Institute of Artificial Intelligence
Ulm University
Ulm, Germany
e-mail: *firstName.lastName*@uni-ulm.de

***Abstract*—We prove a new complexity result for Partial-Order Causal-Link (POCL) planning which shows the hardness of refining a search node (i.e., a partial plan) to a valid solution given a delete effect-free domain model. While the corresponding decision problem is known to be polynomial in state-based search (where search nodes are states), it turns out to be intractable in the POCL setting. Since both of the currently best-informed heuristics for POCL planning are based on delete relaxation, we hope that our result sheds some new light on the problem of designing heuristics for POCL planning.**

Based on this result, we developed a new variant of one of these heuristics which incorporates more information of the current partial plan. We evaluate our heuristic on several domains of the early International Planning Competitions and compare it with other POCL heuristics from the literature.

I. Introduction

Partial-Order Causal-Link (POCL) planning [1], [2] is a technique for solving classical planning problems via search in the space of plans. The currently most prominent approach for solving such problems is planning as search in the space of *states*, although POCL planning has several advantages compared to state-based planning: POCL planning follows a *least commitment principle*, in which only necessary decisions are performed like maintaining only a partial order on the plan steps in a partial plan and maintaining only a partial variable binding thereby avoiding unnecessary restrictions, which might turn out as wrong later in the search. Since solutions are also only partially ordered and all causal dependencies between plan steps are explicitly represented, POCL planning allows greater flexibility at plan execution time [3] and the explanation of the structure of the solution at hand [4].

Despite these advantages, POCL planning has become less attractive to many researchers, because planning systems based on this approach are currently not competitive with current state-of-the-art (state-based) planning systems in terms of runtime. We assume this can be attributed to two main reasons: first, there is a large number of highly informed heuristics for state-based search [5], and, second, there is a lot of theoretical work which helps designing new heuristics or improving existing ones [5], [6]. For example, Bylander's result showing that the plan existence problem given a state and a delete effect-free domain is polynomial [6] lead to the development of several tractable heuristics based on delete relaxation. The probably most prominent one is the FF heuristic as implemented in the Fast Forward (FF) planning system [7]. The success of these heuristically guided systems lead to a paradigm shift from several approaches like POCL planning and CSP-based approaches such as GraphPlan [8] to state-based planning.

We are only aware of two well-informed heuristics for domain-independent, non-temporal POCL planning: The *Relax heuristic* [9] and the *Additive heuristic for POCL planning* [10]. Both heuristics are based on the same idea of finding a solution using a delete-relaxed planning domain. In contrast, heuristics used in state-based search also rely on *critical paths*, *abstractions*, and *landmarks* [5]. We recently developed an approach which enables the use of state-based heuristics in POCL planning, but it did not yet result in a system competitive with state-based planners [11]. We believe that the small number of heuristics for POCL planning can be attributed to the fact that it seems to be much more complicated to estimate the goal distance for a partial plan than to estimate the goal distance for a state. To shed some light on that difficulty, we study the complexity of the plan existence problem for POCL planning, when delete relaxation is performed – analogously to the proposition of Bylander [6] for state-based planning. Based on our results, we improve the *Relax Heuristic* and present empirical results.

The remainder of the paper is structured as follows: Section II is devoted to the formalization of POCL planning. Section III shows our new complexity result. Section IV presents our new heuristic including an empirical evaluation and, finally, the last section concludes the paper.

II. Problem Formalization

Although POCL planning is ordinarily done in a lifted fashion [10], we base our formalization on a fully ground, propositional representation.

A domain model $\langle \mathcal{V}, \mathcal{A} \rangle$ consists of a finite set of propositional state variables $\mathcal{V}$ implicitly defining the induced state space $\mathcal{S} = 2^{\mathcal{V}}$ and a finite set of available actions $\mathcal{A}$. Each action $a = \langle pre, add, del \rangle \in \mathcal{A}$ is a tuple from the set $2^{\mathcal{V}} \times 2^{\mathcal{V}} \times 2^{\mathcal{V}}$ and defines the action's precondition pre, its add list add, and its delete list del. It is applicable in a state $s \in \mathcal{S}$ iff $pre \subseteq s$ and, if applicable in that state, generates the state $(s \setminus del) \cup add$. The applicability and application of action sequences is defined in the straight-forward way.

Partial plans are abstractions of totally ordered action sequences in the sense of actions being only partially ordered; furthermore, partial plans explicitly model effect/precondition relations between different actions by means of so-called causal links. More formally, a partial plan is a tuple $(PS, \prec, CL)$ with PS being a finite set of *plan steps*, $l{:}a \in PS$ consisting of an action $a \in \mathcal{A} \cup \{a_0, a_\infty\}$ and l being a unique label symbol to identify the correct plan step in case the partial plan contains multiple occurrences of a. The two special actions a_0 and a_∞ encode the initial state of the problem and its goal description, respectively. Thus, a_0 has an empty precondition, an empty delete list and the initial state as add effect, and the action a_∞ has an empty add and delete list, and the goal description as precondition. The corresponding plan steps $l_0{:}a_0$ and $l_\infty{:}a_\infty$ are called *init* and *goal*, respectively. The ordering constraints are represented by the strict partial order $\prec$ defined on the plan steps of PS. Every partial plan satisfies $(0, \infty) \in \prec$ and, furthermore, every plan step in PS different from *init* and *goal* is ordered between these two plan steps. The set CL specifies the causal links. A causal link $l \xrightarrow{v} l'$ specifies that the precondition variable $v \in \mathcal{V}$ of the plan step $l'{:}a'$ is provided by the add list of plan step $l{:}a$.

A POCL planning problem is a tuple $\pi = \langle \mathcal{D}, P \rangle$ with $\mathcal{D}$ being a domain model and P being a partial plan. Typically, P contains only the two plan steps *init* and *goal* without causal links, since such a problem corresponds to an ordinary STRIPS planning problem [12], where only an initial state and a goal description is given in addition to the domain model.

A partial plan $P_{sol} = (PS_{sol}, \prec_{sol}, CL_{sol})$ is a solution to a POCL planning problem π, called *plan*, if and only if:

(1) P_{sol} is a refinement of P. That is, if $P = (PS, \prec, CL)$, then $PS \subseteq PS_{sol}$, $\prec \subseteq \prec_{sol}$, and $CL \subseteq CL_{sol}$.

(2) There are no *open preconditions*. That is, for every plan step $l{:}a \in PS_{sol}$, $a = (pre, add, del)$ with $v \in pre$, there is a causal link $l' \xrightarrow{v} l \in CL_{sol}$ with $l'{:}a' \in PS_{sol}$, $a' = (pre', add', del')$, and $v \in add'$.

(3) There are no *causal threats*. That is, if there is a causal link $l \xrightarrow{v} l'' \in CL_{sol}$, then for all plan steps $l'{:}a' \in PS_{sol}$ with $a' = (pre', add', del')$ and $v \in del'$ it holds that the set $\prec_{sol} \cup \{(l, l'), (l', l'')\}$ is no strict partial order.

Criterion (1) relates solutions to the problem specification, i.e., to the initial partial plan P. Criteria (2) and (3) ensure that the solution is *executable* in the sense that any action sequence induced by the ordering constraints is applicable in the initial state and generates a state satisfying the goal condition.

POCL planning procedures perform plan-based search, starting with the initial partial plan and refining it until a solution is generated [10]. To that end, the violation of criteria (2) and (3) is represented by so-called flaws; thus, a partial plan is a solution iff it does not show any flaws. In a first step, a most-promising partial plan is selected from a set of candidates. For that partial plan, first, all its flaws are identified, i.e., all its open preconditions and all causal threats. Then, one of these flaws is selected and resolved using all available possibilities (for instance, an open precondition flaw can be resolved by inserting a causal link rooted either in a plan step from the current partial plan or in a new plan step taken from the domain). All resulting plans are then added to the set of candidates and a new cycle starts over.

This procedure has two decision points: The selection of a most promising partial plan and the selection of a flaw. In this paper, we focus on the former by means of heuristics.

III. Complexity Results

POCL procedures pick a most-promising partial plan from a set of candidates based on some criteria; often, an informed search procedure like A* using a heuristic function is chosen. As noted in the introduction, several well-informed heuristics have been developed for state-based planning [5], but we are only aware of *two* heuristics for POCL planning: the *Relax heuristic* [9] and the *Additive heuristic for POCL planning* [10] (*Add*, for short). Both approximate the number of necessary actions to solve a delete-relaxed version of the planning problem. In state-based planning, ignoring delete lists is a promising idea for constructing heuristics, as the plan existence problem for a delete effect-free domain is known to be solvable in polynomial time [6]. However, in POCL planning, that problem has not yet been investigated in detail.

Let $\pi = \langle \mathcal{D}, P \rangle$ be a POCL planning problem with P being an arbitrary partial plan, i.e., possibly containing more plan steps than just *init* and *goal* and possibility containing causal links and ordering constraints. Such problems are generated during planning, as each search node is a new partial plan and hence induces a new planning problem.

In state-based planning, performing delete-relaxation is straight-forward, as simply all actions in the domain need to be relaxed (although there are more elaborated approaches, which ignore only some of the variables in the delete lists [13]). However, in the POCL setting, in addition to the actions in the domain, we have the actions/plan steps in the current partial plan and the question arises whether these should be relaxed as well. If not, the actions from the domain would still *all* be delete-free, but the plan steps in P are not. We can motivate leaving P unaltered by considering the analog question in state-based planning: there, search nodes are states; hence, every state *completely* reflects the entire information about the search path from the initial state up to the current one. This includes all applied actions leading to that state including their negative effects. In POCL planning, this "planning progress" is reflected in the current partial plan. Relaxing its actions would mean to ignore information about the current progress of the search. Thus, the question we would like to have answered is: "Given P, how hard is it to refine it to a solution given an easier *planning domain*?". In the following, we study the hardness of that problem where this "easier" domain is obtained by performing delete relaxation.

Unfortunately, it turns out that relaxing only the actions in the domain is $\mathbf{NP}{-}\mathbf{complete}$, whereas the problem is in $\mathbf{P}$, if also the actions in P are delete-relaxed. We prove the first result formally, but omit a proof for the latter, since it is trivially solvable in polynomial time in the size of $|\mathcal{D}| + |P|$.

Let us first define our notion of delete relaxation. We call a POCL planning problem $\pi = \langle \mathcal{D}, P \rangle$ with $\mathcal{D} = \langle \mathcal{V}, \mathcal{A} \rangle$ *delete-free*, if and only if for each $(pre, add, del) \in \mathcal{A}$, $del = \emptyset$. We hence call a POCL planning problem π' *delete-relaxed* if it is obtained from a POCL planning problem π by ignoring the delete lists of the actions in $\mathcal{A}$ (but leaving P unaltered).

The decision problem for determining whether a partial plan has a solution using a delete-relaxed domain model is then given by $\mathbf{PLANSAT} := \{\pi | \pi$ is a delete-free POCL planning problem and has a solution$\}$.

Theorem 1. PLANSAT *is* NP−complete*.*

Proof: Membership. Fix an arbitrary delete-free POCL planning problem $\langle \mathcal{D}, P \rangle$ with domain $\mathcal{D} = \langle \mathcal{V}, \mathcal{A} \rangle$ and $P = (PS, \prec, CL)$. Guess a linearization of the plan steps in PS, which respects the ordering constraints of P. We need to show that it can be verified in polynomial time that such a sequence $\mathit{init}, l_1{:}a_1, \dots, l_n{:}a_n, \mathit{goal}$ can be extended to an applicable action sequence using (the delete-free) actions from the domain. This is sufficient for membership, as a POCL solution can be obtained from such a sequence by inserting causal links, which can also be done in polynomial time.

First, we need to verify that the chosen linearization does not violate any causal links present in P. This is the case if and only if it does not have any causal threats, which can be verified in polynomial time. Note that the causal links may only be violated by the plan steps already present, but not by the additional actions from the domain, as these actions do not show delete lists and hence cannot cause new causal threats.

Afterwards, we build a saturated relaxed planning graph [8] starting from *init*. This graph can be built in polynomial time [7]. Furthermore, if the precondition of the plan step $l_1{:}a_1$ is contained in the last fact layer $L \subseteq \mathcal{V}$ of this planning graph, we have verified that we can find a sequence of actions to support the precondition of that plan step (or proved its non-existence, otherwise), since such a sequence can be extracted from the planning graph without backtracking due to the absence of negative effects [7]. Now, we apply that plan step by removing the delete list from L and adding its add list thereby generating a new state. From this state, we build another saturated planning graph to test whether the precondition of the plan step $l_2{:}a_2$ holds in its last fact layer. We repeat that procedure thereby building $|PS| - 1$ planning graphs, one between each tuple of two consecutive plan steps. If the precondition of *goal* is contained in the last layer of the last planning graph, we have verified that the chosen linearization can be extended to an applicable action sequence containing the plan steps of P in an order compatible with $\prec$ and respecting its causal links.

Hardness. To prove the hardness, we adapt a proof by Nebel and Bäckström [14, Theorem 15], in which they proved the NP completeness of deciding whether there exists an applicable action sequence given a partial plan without causal links and without the capability of inserting actions from the domain; i.e., the problem studied was to find a suitable order of the plan steps. We show that this problem does not become easier when one is allowed to insert delete-relaxed actions, independently of whether causal links are present or not. From that observation follows hardness, since P can be *refined*[1] to a POCL solution if and only if there exists a linearization of the plan steps in P which can be extended to an applicable action sequence using actions from the domain.

The proof is done by reduction from CNF-SAT. Given a set of boolean variables $X = \{x^1, \dots, x^n\}$ and a set of clauses $C = \{c^1, \dots, c^m\}$, each clause c^j being a set of literals over X representing a disjunction, we construct a delete-free POCL planning problem, whose solutions are isomorphic to that of the CNF-SAT problem. The general idea is that the initial partial plan (depicted in Fig. 1) contains two plan steps/actions $A_i^\top$ and $A_i^\perp$ for each variable x^i in X. The order in which these plan steps occur in a solution encodes the truth assignment of x^i using the two state variables $x_i{=}\top$ and $x_i{=}\perp$.

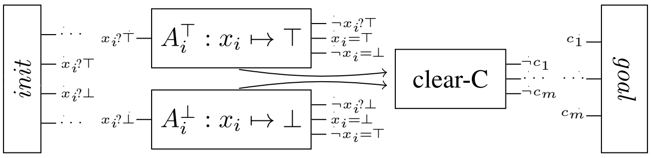


Fig. 1. The initial partial plan used by the hardness proof of Theorem 1.

For example, if $A_i^\top$ appears after $A_i^\perp$ in a solution plan, then $x_i{=}\top$ does hold at the end of any action sequence derived from that plan, and $x_i{=}\perp$ does not. Note that every pair of two actions $A_i^\top$ and $A_i^\perp$ *needs* to be ordered w.r.t. each other, since one plan step would *threaten* the other as soon as a causal link is set over $x_i{=}\top$ or $x_i{=}\perp$, respectively.

Furthermore, adding the delete-relaxed variants of these actions does not change which of these variables finally holds for the following reasons: First, the delete-relaxed variant of $A_i^\top$, for example, cannot be applicable *after* $A_i^\top$ due its precondition and delete effect $x_i?\top$, which encodes whether this action has already been executed. Second, inserting it *before* $A_i^\top$ does not change which of the variables $x_i{=}\top$ and $x_i{=}\perp$ finally holds, because only the last applied *non-relaxed* action determines the final outcome.

We still need to "use" these variables in order to determine whether the resulting variable assignment satisfies the SAT formula. To that end, the domain model contains one action for each literal in any clause which serves the purpose of setting a clause to true if one of its literals is true. Thus, if $x^i \in c^j$ and $c^j \in C$, then $C_{ij}^\top := (\{x_i{=}\top\}, \{c_j\}, \emptyset) \in \mathcal{A}$. The action $C_{ij}^\perp$ is defined analogously for $\neg x^i \in c^j$, $c^j \in C$. Obviously, these actions can be used to support the preconditions of the *goal* plan step (and thus solve the SAT formula) if and only if a satisfying variable assignment was chosen, i.e., if a correct order of the actions $A_i^\top$ and $A_i^\perp$ was found. However, since these actions may be inserted at an arbitrary position in the partial plan, and in every solution $x_i{=}\top$ and $x_i{=}\perp$ (which are the preconditions of $C_{ij}^\top$ and $C_{ij}^\perp$, respectively) are both true at some point, we need to ensure that the goal's preconditions are only supported by those actions $C_{ij}^\perp$ and $C_{ij}^\top$, which were applied after the very last action from $A := \{A_i^\top, A_i^\perp | i \in \{1, \dots, n\}\}$. We ensure this by means of the action clear-C, which is ordered after the ones from A and deletes all state variables c_j. Since the actions $C_{ij}^\top$ and $C_{ij}^\perp$ are already delete-relaxed and the delete relaxation of clear-C is a no-op, there are no further cases to consider.

See Appendix A for the formal problem specification. ■

The main result of our theorem is that performing delete-relaxation only for the domain's actions is not sufficient to obtain a tractable problem class. Looking closer to the proof reveals that the complexity lies in finding the correct order of the plan steps. The possibility to insert delete-free actions into a partial plan does not make this problem easier.

[1] Please note that this question is different from verifying that P *already is* a valid solution, which can be done in polynomial time using the POCL solution criteria [14, Theorem 14].

IV. Sample-FF

In this section, we introduce *Sample-FF*, a heuristic greatly inspired by the constructive proof of the $\mathbf{NP}$ membership proof of Theorem 1 presented in the last section. *Sample-FF* is based on the same ideas as the *Relax heuristic* [9], which is an adaptation of the FF heuristic [7] for state-based planning. However, *Sample-FF* incorporates more information about the currently considered partial plan. In particular, it does not ignore the negative effects of its plan steps and is able to use the constraints implied by the causal links. Before we explain how *Sample-FF* works in detail, we briefly review the *Relax heuristic* to pinpoint the differences.

Given a partial plan P, the *Relax heuristic* estimates the distance from the initial state (i.e., the postcondition of *init*) to an "artificial" goal description, which is obtained by building the union of all open preconditions of the plan steps of P. This goal distance is obtained in the same way the FF heuristic estimates the distance from a state to the goal description: it solves a delete-relaxed version of the problem and uses the number of actions of that solution as heuristic estimate[2]. Thus, the *Relax heuristic* can be calculated very efficiently, since the problems solved are in $\mathbf{P}$. However, while tractable on the one hand, the performed relaxation is quite severe on the other: The heuristic uses only the set of open preconditions of the plan steps in P, thereby ignoring their quantity, their negative effects, and the constraints posed by the causal links.

However, although delete-relaxed planning with a non-relaxed partial plan is $\mathbf{NP}-\mathbf{complete}$, there is no need to ignore the negative effects and the causal links altogether. In particular the causal links are of interest, since even a single wrongly placed causal link can prevent that the respective partial plan may be refined to a valid solution. Looking at the $\mathbf{NP}$ membership part of the proof of Theorem 1 though, it becomes clear that the "hard" part of extracting a solution is only guessing a suitable linearization. Given such a linearization, the rest of the proof can be directly translated into a deterministic *polytime* program consisting of the following three phases.

First, we generate some linearizations of the partial plan at hand. This is done using sampling and simulates the "guessing" part of the $\mathbf{NP}$ membership proof.

Second, we estimate the cost of completing the linearizations sampled in the first phase into relaxed solution plans. We look at each linearization separately and for each create a sequence of relaxed planning graphs that reflect the argumentation in the membership part of the proof of Theorem 1. The number of required additional relaxed actions inserted in this process yields a heuristic estimate for each linearization.

Last, we derive a heuristic estimate for a partial plan by combining heuristic estimates for its linearizations, i.e., by taking the minimum of the computed linearization estimates. We will next take a closer look at the individual phases and some possible optimizations of the basic idea.

A. Sampling Linearizations

Since it is infeasible to consider *all* linearizations of a given partial plan, we need to limit the number of considered linearizations to receive a practical heuristic, e.g., by a constant. We also want to avoid choosing linearizations that are too similar to each other to avoid introducing too much bias into the heuristic. Therefore, we use a Markov Chain Monte Carlo approach for approximately *uniformly* sampling a constant number of linearizations [15].

We define a *linearization graph* whose nodes are linearizations consistent with the partial order. There is an edge between two nodes when their corresponding linearizations can be converted into each other by swapping two adjacent plan steps. Performing a random walk in the linearization graph thus corresponds to a sequence of swappings thereby generating a new linearization consistent with the ordering constraints. Let $d(z)$ be the degree of the node representing a linearization z of the plan steps PS in the linearization graph. Thus, it holds $d(z) \leq |PS|-1$. We define the probability of proceeding from a linearization z to its neighbor z' as follows:

$$p(z, z') = \begin{cases} \frac{1}{2(|PS|-1)} & \text{if } z \text{ and } z' \text{ are adjacent,} \\ 1 - \frac{d(z)}{2(|PS|-1)} & \text{if } z = z', \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the fewer neighbors a linearization has (i.e., the lower the number of swaps consistent with the underlying partial order), the more likely staying at the current linearization becomes. It can be shown that a uniform distribution over all linearizations is reached after polynomially many random walks. By choosing an arbitrary consistent initial linearization and doing enough random walks, we can thus uniformly sample from the set of possible linearizations.

B. Estimating the Cost for Linearizations

Let $z = \mathit{init}, l_1{:}a_1, \ldots, l_n{:}a_n, \mathit{goal}$ be a sampled linearization of a partial plan which is not yet a solution. This linearization represents a totally ordered partial plan with "gaps" where some steps are still missing. Filling the gaps with appropriate non-relaxed plan steps will create a solution plan. Therefore, the number of relaxed plan steps required for filling the gaps can serve as a heuristic estimate for that linearization. We begin by building a saturated planning graph starting at the postcondition of *init*. We then apply $l_1{:}a_1$ in its last fact layer, yielding a "state" s_1. If $l_1{:}a_1$ cannot be applied in the last fact layer, the linearization can be discarded, since it cannot be completed into a solution. Otherwise a new saturated planning graph is constructed starting in s_1, in whose last layer $l_2{:}a_2$ is applied, yielding s_2, and so on. The last such planning graph is built after applying $l_n{:}a_n$. When *goal* is applicable in the last fact layer of the last planning graph, we know that a relaxed solution exists and we can proceed with extracting it.

For this, we traverse the constructed planning graphs last to first. For the last graph, this amounts to doing standard relaxed solution extraction in the same fashion of *Relax*. Then, the non-relaxed plan step $l_n{:}a_n$ is applied in reverse, the second-last graph is considered, and so on until the postcondition of *init* is reached. The heuristic value for z is then defined as the total number of relaxed plan steps required in all constructed planning graphs.

[2] More precisely, only a subset of these actions is used: If an action in the delete-relaxed solution corresponds to a (non-relaxed) plan step in the partial plan, it does not count towards the cost estimate.

C. Combining Estimates for Linearizations

The last phase is conceptually simple: take the minimum of all estimates for the sampled linearizations.

An important corner case, however, is the situation where none of the sampled linearizations can be completed to a relaxed solution, because then the minimum of all estimates does not represent a finite heuristic value. For other heuristics, an infinite heuristic value does not pose a problem: It usually means that a partial plan cannot be completed to a relaxed solution, let alone a real solution, and can therefore be safely discarded. In our setting, however, we cannot be sure of this. It might just be coincidence that none of the sampled linearizations could be completed to a relaxed solution, and that the right linearization was missed in the sampling phase. The question which value should be returned in this case is hence not trivial. As pointed out, returning a high value might be too pessimistic and, contrarily, returning zero might be too optimistic; in fact, the partial plan might even be doomed to become invalid, but given the non-exhaustive number of samples, the heuristic was not able to prove that. Hence, we choose a compromise and return the number of open preconditions as estimate. Alternatively, we could return the value of the *Relax heuristic* instead, but we did not evaluate that variant.

D. Optimizations.

In the following, we present a few directions in which the basic algorithm can be improved.

a) Enumerating all linearizations: In cases where the number of linearizations is small, enumerating all of them is desirable, as it yields more accurate heuristic values than sampling. Additionally, this allows for safe pruning, as the corner case described before cannot occur: A partial plan *can* be discarded when all possible linearizations are proved unsolvable in the relaxed setting, i.e., completeness is guaranteed.

We therefore want an estimate on the number of linearizations a partial plan has and use it to decide whether it is deemed feasible to look at all linearization for it. Unfortunately, determining the exact number of linearizations is $\#\mathbf{P}-\mathbf{complete}$ [15], i.e., hard in the sense that there is no known method substantially better than enumerating all possible linearizations. Hence, we take the direct approach and start enumerating linearizations until we have reached a predefined maximum number of linearizations. If we have not enumerated all linearizations at this point, we switch to sampling, throwing away the linearizations enumerated thus far. Experiments performed in a pre-evaluation indicate that the benefits in precision and pruning power outweigh the effort wasted for generating unused linearizations.

The impact of that optimization heavily depends on the chosen flaw selection function. For example, always preferring "old" flaws, i.e., flaws detected early in the given partial plan, will produce partial plans with a high number of linearizations, since new plan steps are inserted level-wise starting from *goal*, as the oldest flaws in the initial partial plan are the open precondition flaws of *goal* (given that plan contains only the representatives of the initial state and goal description, but no other initial plan steps). If the converse strategy is applied, i.e., if always a flaw is preferred that was detected last, the number of linearizations stays constantly 1 until a sequence of actions has been found, which supports at least one state variable of the goal description and roots in the initial state. Thus, up to this point, enumerating $n \geq 1$ linearizations will exhaustively enumerate all possible linearizations.

b) Precomputing a fixed point for the initial state: Since the forward phase of computing a heuristic value for a linearization always starts at the initial state, the fixed point reached by applying relaxed actions will always be the same. This first fixed point can thus be precomputed once and be reused each time the cost of a linearization is estimated. Note that it is also likely that the extracted solutions contain more relaxed steps before the first non-relaxed step than between two later non-relaxed steps, because typically only a few facts are deleted by applying a non-relaxed step. This makes precomputing a fixed point for the initial state an attractive idea for optimization.

c) Respecting causal links: We can also take a closer look at the relaxed solutions generated for a given linearization. It then becomes apparent that in many cases, these solutions contain relaxed plan steps at places where the POCL planning algorithm would not put non-relaxed plan steps due to the presence of causal links. To illustrate this, let P be a partial plan that contains a causal link $l \xrightarrow{v} l'$ between l and l' over variable v. Let furthermore a be an action whose delete list contains v. Adding a plan step $l''{:}a$ to P creates a causal threat if l'' can be ordered between l and l' and thus prevents P from being a solution. On the other hand, the relaxed version of a can of course be inserted between l and l' when a relaxed solution is constructed. When this happens, the heuristic value is a poor estimate of the true remaining search effort, as it is certain that the relaxed solution conflicts with every potential solution plan that can be generated from P.

We therefore modify the planning graph generation to respect causal links. Let $z = l_0{:}a_0, \ldots, l_{n+1}{:}a_{n+1}$ with $l_0{:}a_0 = \mathit{init}$ and $l_{n+1}{:}a_{n+1} = l_\infty{:}a_\infty = \mathit{goal}$ be a linearization of the plan steps of the partial plan $P = (PS, \prec, CL)$. We define the *active* causal links between l_i and l_{i+1} to be the set $\{l_n \xrightarrow{v} l_m \in CL \mid n \leq i \text{ and } m \geq i+1\}$, i.e., the causal links whose arcs cross an imaginary line drawn between l_i and l_{i+1} in a graphical representation of z. The set of active causal links contains exactly the causal links that can potentially cause a causal threat when an action is put between l_i and l_{i+1}. To be more precise, an action put between l_i and l_{i+1} will lead to a causal threat if its delete list contains a variable mentioned in an active causal link. Such actions are called *threatening*, and we modify relaxed planning graph generation to not use threatening actions. Identifying the active causal links and filtering out actions which are threatening them can obviously be done in polynomial time.

In summary, the improvement works by calculating the set of threating actions each time before the relaxed planning graph between two non-relaxed plan steps is built, and using only non-threatening actions for building the relaxed planning graph. Unfortunately, this optimization can not be used in conjunction with *precomputing* the first fixed point: The causal links that begin in *init* can change between linearizations for different partial plans, and so can the computed fixed point if only non-threatening actions are used. Our implementation can therefore incorporate the causal links rooting in $init$ independently of the remaining causal links, s.t. one can choose between the per-node runtime-improvement obtained by precalculation of the first saturated planning graph versus more informed heuristic values while still being able to incorporate the remaining causal links independently of that choice.

d) Reusing parent linearizations: In our early experiments (before we implemented the following optimization), we observed cases where linearizations with an estimated cost of zero were found for some partial plan visited during search, yet the planner was unable to generate a solution. This is an undesirable and strange situation, since such a linearization can easily be transformed into a POCL solution: Applying the actions of the zero-cost linearization starting in the initial state generates a state satisfying the goal description; that is, there is no need to insert any additional action, neither relaxed nor non-relaxed, only missing causal links and ordering constraints need to be inserted, which can be done very efficiently.

The problem of that situation is the randomized nature of the heuristic. Since in each node a fixed number of samples is generated independently of the linearizations obtained by its parent node, heuristic values may strongly vary between each two consecutive partial plans. To obtain more "stable" heuristic estimates, a partial plan P tries to reuse the best linearization of its parent, "best" being defined as the first linearization for which the smallest heuristic value was obtained. Whether such a parent linearization z can be reused depends on the last applied modification: In case of an insertion of an ordering constraint or a causal link, it must be tested whether z is compatible with the inserted ordering constraints. If it is, $n+1$ samples are used with n being the predefined fixed number of samples, otherwise just n (new) samples are considered. In case of an action insertion, the linearization z is extended to a linearization of size $|z|+1$ by inserting the new action at an arbitrary suitable position. Since z proved being successful for the parent node, we do not only create one reused linearization, but three – with randomly chosen positions for the new action. Thus, in general, each partial plan uses $n+m$ samples with n new sampled linearizations and $m \in \{0,\ldots,3\}$ linearizations obtained from the best linearization of its parent node.

The described improvement "stabilizes" heuristic estimates, since good linearizations remain being used for heuristic estimation. Furthermore, it solves the problem concerning the zero-cost linearizations: Since the POCL algorithm is complete and zero-cost linearizations are applicable in the initial state and satisfy the goal condition (otherwise, it would not have cost zero), at least one modification m^* compatible with that linearization must exist for each remaining flaw. Thus, reusing that linearization to compute the heuristic value for the child plan created by applying m^*, the child plan will in turn have a heuristic value of zero; hence, after a partial plan with heuristic zero is found, a solution is obtained shortly afterwards. Note that this does not mean that the planner will return the linearization itself as a solution. Since only a sufficiently small number of causal links and ordering constraints is added in order to receive a solution, the least-commitment principle of POCL planning is preserved.

E. Evaluation

We implemented the proposed heuristic within our POCL planner, which is implemented in Java®. As search strategy, we used weighted A* using a weight of 2. That is, in each cycle, a partial plan p is selected with minimal f value, f given by $f(p) = g(p) + 2 * h(p)$, g being the unit cost of the partial plan and h being its heuristic estimate. In case two partial plans have the same f value, we break ties by preferring a partial plan with higher cost, thereby preferring smaller heuristic values. Remaining ties are broken using the $LIFO$ strategy thereby preferring newest partial plans. Concerning the flaw selection strategy, we always select a newest flaw, where all flaws detected in the same partial plan are regarded equally new/old. Among these flaws, we break ties by pursuing the *Least Cost Flaw Repair* selection [16], which prefers a flaw for which there are the least number of modifications. Remaining ties are broken by chance.

We compare the *Sample-FF heuristic* with the two currently best-informed heuristics for POCL planning: the *Relax heuristic* [9] and the *Additive heuristic for POCL planning* [10] (*Add*, for short). In addition to these heuristics from the literature, we implemented a new variant of the *Relax heuristic*. It only differs from the original version by a small detail: *All* actions of a relaxed solution count towards the heuristic estimate, whereas the original version ignores actions, which already exist in the given partial plan. This variant, called *Relax** dominates the original version while being "more inadmissible". Performance is measured in terms of size of the produced search space and number of solved problem instances based on several benchmarks taken from the early International Planning Competitions (IPCs).

The evaluated domains and problems are taken from the IPC 1 to IPC 5 (cf. Tab. I). For each domain, we used n consecutive problem instances, starting with the smallest ones. We omitted domains for which all configurations timed out on all problem instances. We used a time limit of 15 minutes CPU time and a memory limit of 2 GB. We run our experiments on a machine with two Intel Xeon® processors, each having 8 physical cores running at 2,6 GHz.

When comparing the performance of *Relax*, *Relax**, and *Sample-FF* with *Add* in terms of solved problems in total, we see that *Add* clearly dominates all other heuristics. This comes to our very surprise, as *Add* may heavily overestimate the optimal relaxed goal distance. In fact, *Relax* can be regarded as an *improvement* over *Add*, which avoids this overestimation.

As opposed to the other evaluated heuristics, *Sample-FF* has parameters which need to be specified. To achieve a polytime-bounded procedure, we fix the number of sampled linearizations to a predefined constant. We evaluated 1, 3, 10, and 30 samples per search node. The more samples are chosen, the more accurate the heuristic becomes. However, clearly, the calculation time becomes much more expensive as it scales linearly with the number of samples. The optimizations of trying to enumerate all linearizations and to reuse parent linearizations are always turned on, as we observed a clear improvement in terms of solved problem instances in a small pre-evaluation. Concerning respecting causal links, the heuristic features to respect no causal links at all, all causal links, or just the ones which are not rooting in $init$ (as was motivated in

TABLE I. THIS TABLE COMPARES THE DIFFERENTLY PARAMETRIZED VERSIONS OF *Sample-FF*. "*front:*" AND "*end:*" SPECIFY WHETHER CAUSAL LINKS ROOTING IN $init$ (OR *not* ROOTING IN $init$, RESPECTIVELY) WERE USED TO REDUCE THE SET OF APPLICABLE ACTIONS IN THE RESPECTIVE LAYERS OF THE SAMPLED LINEARIZATIONS. THE DOMAINS ARE ORDERED BY THE IPC, IN WHICH THEY WERE FIRST USED. THE NUMBER n SPECIFIES THE NUMBER OF USED PLANNING PROBLEMS IN THE RESPECTIVE DOMAIN. THE ENTRIES SPECIFY THE NUMBER OF SOLVED INSTANCES OF THE RESPECTIVE DOMAIN. BOLD ENTRIES SPECIFY THE CONFIGURATION WITH THE LARGEST NUMBER AMONG ALL CONFIGURATIONS OF *Sample-FF*.

					Sample-FF											
Domain	n	**Add**	**Relax***	**Relax**	front: $\perp$ end: $\perp$				front: $\perp$ end: $\top$				front: $\top$ end: $\top$			
					1	3	10	30	1	3	10	30	1	3	10	30
grid	5	0	0	0	0	0	0	0	0	0	0	0	1	**1**	**1**	**1**
gripper	20	14	20	7	1	1	1	1	1	2	1	1	2	**3**	**3**	2
logistics	20	12	8	7	**8**	5	6	6	6	7	6	5	0	0	1	1
movie	30	30	30	30	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
mystery	20	8	8	9	10	11	9	9	10	11	10	9	**12**	**12**	11	11
mystery-prime	20	3	3	3	3	4	**6**	5	5	4	4	4	**6**	**6**	**6**	**6**
blocks	21	4	5	7	5	5	**6**	**6**	4	3	3	3	5	3	2	0
logistics	28	28	28	27	22	23	23	**24**	21	19	20	21	15	13	14	15
miconic	100	100	49	39	**40**	**40**	37	35	39	41	37	32	15	16	18	20
depot	22	2	2	1	1	1	1	1	0	1	1	1	2	2	**3**	2
driverlog	20	7	9	7	**11**	9	10	9	9	10	9	8	8	7	9	7
rover	20	20	18	19	13	14	**15**	**15**	11	11	12	11	7	9	9	9
zeno-travel	10	4	5	3	3	**5**	4	**5**	4	3	4	4	1	1	1	1
airport	20	18	15	9	10	**11**	**11**	**11**	7	10	10	10	7	8	6	4
pipesworld-noTankage	10	8	1	2	2	3	**5**	3	2	1	2	1	1	4	3	**5**
pipesworld-Tankage	10	1	1	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
satellite	20	16	7	7	**7**	5	6	6	5	5	7	5	1	2	3	3
pipesworld	10	1	1	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
storage	20	7	6	4	6	7	9	8	6	7	7	6	9	9	**10**	**10**
tpp	20	19	11	11	**8**	7	6	7	6	6	6	6	5	6	6	6
total	446	292	227	194	182	183	187	183	168	173	171	159	127	132	136	133

the last section). Again, we obtain a trade-off between accuracy and calculation speed. While respecting all causal links is the most informed variant, it is clearly the slowest one, as it has to calculate the set of applicable actions between each two plan steps in a sampled linearization, as explained in the last section. We evaluated all possible combinations of the two parameters leading to 12 variants of *Sample-FF*.

Investigating the use of causal links, our experiments seem to suggest that the additional overhead incurred by respecting them does not pay off in total: We clearly see that the variant which does not respect causal links dominates the other two configurations of *Sample-FF*. While this is true taking into account the total number of solved problems, there are also domains where the configuration respecting all causal links solved more problems than *all other* heuristics. In one particular unsolvable problem instance, *all Sample-FF* configurations respecting all causal links were able to prove that problem to be unsolvable, whereas all other heuristics including *Relax*, *Relax** and *Add* incurred time-outs. Concerning the optimal number of samples, there is not a clear result, but we can observe that an optimal choice lies between 1 and 10.

Taking a look at the overall number of solved problems, the best configuration is the one which does not respect causal links and uses 10 samples. That configuration is clearly competitive with *Relax* in terms of the number of solved problems, as it solved 187 out of 446 problems, whereas *Relax* solved 194 problems. Investigating the number of domains in which one heuristic performed better than another, the data reveals that the configuration using no causal links and just a single sample is even better than *Relax*. In 6 out of 20 domains it solved strictly more problems than *Relax*, whereas *Relax* strictly dominated *Sample-FF* in only 5 domains. While it seems discouraging that using causal links seems not to pay off, we see a potential in further improving the heuristic: We evaluated the ratio of created partial plans to the ones for which no heuristic value could be obtained due to infeasible samples. As stated in the previous section, we return the number of open preconditions in these cases to prevent being blind. That is, a ratio of 100% would correspond to a heuristic which is *entirely* based on the number of open preconditions. We discovered that the mean ratio ranges from 4% to 1% (for increasing number of samples) for the configurations which do not respect causal links, from 7% to 3% for the configurations using only causal links not rooting in $init$, and from astonishing 46% to 36% for the configurations which respect all causal links. This clearly indicates the impact of the constraints posed by the causal links, as even for 30 samples, in 36% of all created nodes there does not exist a delete-relaxed solution. While this proves our assumption correct that causal links have a major impact on the set of valid solutions which can be derived from partial plans, we still need to find a way how to cope with these cases. Note that it might also be that in many of these cases the corresponding partial plans could actually have been pruned from the search space given *all* linearizations but we cannot decide this being the case in polynomial time.

Considering only the number of solved problem instances does not tell how well-informed the respective heuristics are. A larger number may also be attributed to the time a heuristic needs to be evaluated, as faster heuristics allow for a larger search space within the time limit. We hence investigated the number of solved problem instances given the number of generated search nodes. Heuristics which have a larger number of solved instances given the same number of generated search nodes can thus be regarded more accurate. Our data reveals that adding more samples improves heuristic accuracy and that the variant without respecting causal links is the most informed one among all *Sample-FF* configurations. However, the last result can be attributed to the large number of partial plans for which no heuristic value could be calculated. If we figure out how to solve this problem, the variant respecting causal links will probably improve its performance significantly.

V. Conclusion

In this paper, we made two contributions to the field of POCL planning: We proved that the plan existence problem given a search node in standard POCL planning (i.e., an arbitrary non-relaxed partial plan) and a delete-relaxed planning domain is $\mathbf{NP}-\mathbf{complete}$. This is an interesting observation, since the corresponding decision problem in state-based planning is in $\mathbf{P}$. Based on the constructive proof of our main complexity result, we developed a new heuristic for POCL planning and presented empirical results.

The presented heuristic can still be improved. In particular, we want to solve the problem that for many plans no plan step sequence was found that could be extended to a relaxed solution. Also, we want to adapt our heuristic to lifted planning, s.t. it can evaluate partial plans which are not fully ground.

Appendix
Problem Formalization of Hardness Proof

Given a CNF-SAT problem (i.e., a SAT formula given in conjunctive normal form) with variables $X = \{x^1, \ldots, x^n\}$ and a set of clauses $C = \{c^1, \ldots, c^m\}$, we construct the delete-free POCL planning problem $\pi = \langle \mathcal{D}, P \rangle$ with $\mathcal{D} = \langle \mathcal{V}, \mathcal{A} \rangle$:

$$\begin{aligned}
\mathcal{V} = {} & \{x_i?\top, x_i?\bot, x_i{=}\top, x_i{=}\bot \mid i \in \{1, \ldots, n\}\} \cup \\
& \{c_i \mid i \in \{1, \ldots, m\}\} \\
\mathcal{A} = {} & \{\textit{dr-}A_i^\top, \textit{dr-}A_i^\bot \mid i \in \{1, \ldots, n\}\} \cup \{\textit{dr}\text{-clear-C}\} \cup \\
& \{C_{ij}^\top \mid c^j \in C, x^i \in c^j\} \cup \{C_{ij}^\bot \mid c^j \in C, \neg x^i \in c^j\} \\
P = {} & (PS, \prec, CL) \text{ and} \\
PS = {} & \{l_0{:}a_0, l_\infty{:}a_\infty, l_C{:}\text{clear-C}\} \cup \\
& \{l_i^\top{:}A_i^\top, l_i^\bot{:}A_i^\bot \mid i \in \{1, \ldots, n\}\} \\
\prec = {} & \{(l_i^\top, l_C), (l_i^\bot, l_C) \mid i \in \{1, \ldots, n\}\} \\
CL = {} & \emptyset
\end{aligned}$$

The actions are given as follows:

$$\begin{aligned}
a_0 &= (\emptyset, \{x_i?\top, x_i?\bot \mid i \in \{1, \ldots, n\}\}, \emptyset) \\
a_\infty &= (\{c_i \mid i \in \{1, \ldots, m\}\}, \emptyset, \emptyset) \\
A_i^\top &= (\{x_i?\top\}, \{x_i{=}\top\}, \{x_i?\top, x_i{=}\bot\}) \\
\textit{dr-}A_i^\top &= (\{x_i?\top\}, \{x_i{=}\top\}, \emptyset) \\
A_i^\bot &= (\{x_i?\bot\}, \{x_i{=}\bot\}, \{x_i?\bot, x_i{=}\top\}) \\
\textit{dr-}A_i^\bot &= (\{x_i?\bot\}, \{x_i{=}\bot\}, \emptyset) \\
\text{clear-C} &= (\emptyset, \emptyset, \{c_1, \ldots, c_m\}) \\
\textit{dr}\text{-clear-C} &= (\emptyset, \emptyset, \emptyset) \\
C_{ij}^\top &= (\{x_i{=}\top\}, \{c_j\}, \emptyset) \\
C_{ij}^\bot &= (\{x_i{=}\bot\}, \{c_j\}, \emptyset)
\end{aligned}$$

Acknowledgment

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

[1] D. McAllester and D. Rosenblitt, "Systematic nonlinear planning," in *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991)*. AAAI Press, 1991, pp. 634–639.

[2] J. S. Penberthy and D. S. Weld, "UCPOP: A sound, complete, partial order planner for ADL," in *Proceedings of the third International Conference on Knowledge Representation and Reasoning*. Morgan Kaufmann, 1992, pp. 103–114.

[3] C. Muise, S. A. McIlraith, and J. C. Beck, "Monitoring the execution of partial-order plans via regression," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. AAAI Press, 2011, pp. 1975–1982.

[4] B. Seegebarth, F. Müller, B. Schattenberg, and S. Biundo, "Making hybrid plans more clear to human users – a formal approach for generating sound explanations," in *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press, 6 2012, pp. 225–233.

[5] M. Helmert and C. Domshlak, "Landmarks, critical paths and abstractions: What's the difference anyway?" in *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, vol. 9, 2009, pp. 162–169.

[6] T. Bylander, "The computational complexity of propositional STRIPS planning," *Artificial Intelligence*, vol. 94, no. 1-2, pp. 165–204, 1994.

[7] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research (JAIR)*, vol. 14, pp. 253–302, May 2001.

[8] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, pp. 281–300, 1997.

[9] X. Nguyen and S. Kambhampati, "Reviving partial order planning," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*. Morgan Kaufmann, 2001, pp. 459–466.

[10] H. L. S. Younes and R. G. Simmons, "VHPOP: Versatile heuristic partial order planner," *Journal of Artificial Intelligence Research (JAIR)*, vol. 20, pp. 405–430, 2003.

[11] P. Bercher, T. Geier, and S. Biundo, "Using state-based planning heuristics for partial-order causal-link planning," in *Advances in Artificial Intelligence, Proceedings of the 36nd German Conference on Artificial Intelligence (KI 2013)*, 2013, pp. 1–12.

[12] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.

[13] M. Katz, J. Hoffmann, and C. Domshlak, "Who said we need to relax all variables?" in *Proceedings of the 23d International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 2013.

[14] B. Nebel and C. Bäckström, "On the computational complexity of temporal projection, planning, and plan validation," *Artificial Intelligence*, vol. 66, no. 1, pp. 125–160, 1994.

[15] G. Brightwell and P. Winkler, "Counting linear extensions," *Order*, vol. 8, no. 3, pp. 225–242, 1991. [Online]. Available: http://dx.doi.org/10.1007/BF00383444

[16] D. Joslin and M. E. Pollack, "Least-cost flaw repair: A plan refinement strategy for partial-order planning," in *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*. AAAI Press, 1994, pp. 1004–1009.

The following pages show the publication:

T. Geier and P. Bercher. “On the Decidability of HTN Planning with Task Insertion”. In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011).* AAAI Press, 2011, pp. 1955–1961

Reprinted with kind permission of AAAI Press.

On the Decidability of HTN Planning with Task Insertion

Thomas Geier and **Pascal Bercher**
Ulm University, Institute of Artificial Intelligence, Ulm, Germany
forename.surname@uni-ulm.de

Abstract

The field of deterministic AI planning can roughly be divided into two approaches — classical state-based planning and hierarchical task network (HTN) planning. The plan existence problem of the former is known to be decidable while it has been proved undecidable for the latter. When extending HTN planning by allowing the unrestricted insertion of tasks and ordering constraints, one obtains a form of planning which is often referred to as "hybrid planning".

We present a simplified formalization of HTN planning with and without task insertion. We show that the plan existence problem is undecidable for the HTN setting without task insertion and that it becomes decidable when allowing task insertion. In the course of the proof, we obtain an upper complexity bound of $\mathbf{EXPSPACE}$ for the plan existence problem for propositional HTN planning with task insertion.

1 Introduction

Planning is known to be intractable in general. This holds both for classical state-based planning and for hierarchical planning. For the former, the complexity of the plan existence problem ("Is there a plan that solves the given planning problem?") reaches from constant time over $\mathbf{EXPSPACE{-}complete}$ to undecidable [Erol *et al.*, 1995] depending on various restrictions. The most expressive "configuration" of classical planning that is still decidable[1] is thus intractable. Hierarchical task network (HTN) planning was created to mitigate this complexity problem by introducing a hierarchy over operators and thereby including search control knowledge into the planning process. Contrary to the intuition that the introduced hierarchy makes planning easier, it introduced undecidability even for the case in which classical planning is "only" $\mathbf{PSPACE{-}complete}$[2]; this result was proved by Erol *et al.* [1996] for their formalization of HTN planning [Erol *et al.*, 1994].

The hierarchical aspect in HTN planning is achieved by introducing the concept of *compound tasks* in addition to the operators known from classical planning, which we call *primitive tasks* in this context. Compound tasks can be decomposed into predefined partial plans using so-called decomposition methods. The goal is to find an executable plan that was obtained by decomposing compound tasks in the initial plan until all tasks are primitive. Although the provision of decomposition methods can speed up the search process, they introduced undecidability. However, several restrictions can be imposed on methods which make the plan existence problem decidable: for instance if the initial task network and all methods are totally ordered or if all methods are acyclic [Erol *et al.*, 1996]. In this work, we show how decidability can be achieved without restricting methods by altering the solution criterion of HTN planning problems.

We investigate the decidability of the plan existence problem for *HTN planning with task insertion*, a planning formalism that lies between classical and HTN planning and is therefore often referred to as *hybrid planning*: the idea is to have an initial task network that needs to be decomposed, but to allow the insertion of tasks without requiring them to be inserted by the decomposition of compound tasks. This particular variant of fusing classical with HTN planning has been addressed before by Estlin *et al.* [1997], Schattenberg and Biundo [2006], and Gerevini *et al.* [2008]. The most important argument for this planning paradigm is that it overcomes a main criticism of HTN planning: *"HTN planning reduces the flexibility of an agent to respond to situations that are not anticipated by the writer of the task reduction schemas"* [Kambhampati *et al.*, 1998]. This argument is also the main motivation for Kambhampati *et al.*'s formalization of hybrid planning. There, compound tasks also show preconditions and effects like primitive ones and they can be inserted like operators in classical planning (which is also true for the work of Schattenberg and Biundo [2006]). However, Kambhampati *et al.* do not specify an initial task network. Hence, there is *no need* to insert and decompose compound tasks and therefore, they are only an efficiency boost for solving *classical* planning problems.

In the following sections we will first introduce our formalization of HTN planning with task insertion. We explain

[1] Datalog (no function symbols and finitely many constant symbols); operators are given in the input; allow delete lists and negated preconditions [Erol *et al.*, 1995].

[2] Propositional; operators are given in the input; allow delete lists and negated preconditions [Erol *et al.*, 1995].

why our formalism, although stripped of many technical aspects, is still susceptible to the original proof of undecidability for HTN planning when disallowing insertion of tasks. We then present a result that yields an upper bound on the length of shortest solutions and derive from it an upper complexity bound for the plan existence decision problem. In the end we conclude our paper with a discussion of the presented results.

2 Formalism

In this section, we introduce our notion of *HTN planning with task insertion* (or *hybrid planning*, for short). For the sake of simplicity, we base our formalism on a ground state representation. That is, there exist only boolean propositions. The choice effects only the final complexity class but not the decidability, as long as a potential first-order representation does not introduce undecidability on its own, e.g., by the use of functions.

During the rest of the paper, we consider relations and functions as sets of tuples. We use the bar notation for restricting relations and functions. Given a set of tuples $Q \subseteq R \times R$, we define $Q|_X := \{(q_1, q_2) \in Q \,|\, q_1, q_2 \in X\}$ and for a function $f : R \to S$, we define $f|_X := \{(r, s) \in f \,|\, r \in X\}$.

We begin by describing a task network, which represents a generalization of a task sequence in that it is only partially ordered.

Definition 1 (Task Network)**.** A *task network* $\mathrm{tn} = (T, \prec, \alpha)$ over a set of task names X is a tuple, where

- T is a finite and non-empty set of tasks
- $\prec\ \subseteq T \times T$ is a strict partial order on T (irreflexive, asymmetric, and transitive)
- $\alpha : T \to X$ labels every task with a task name

We write TN_X for the set of all task networks over the task names in X. Given a task network $\mathrm{tn} = (T, \prec, \alpha)$ and a set of tasks T', we define its restriction $\mathrm{tn}|_{T'} := (T \cap T', \prec|_{T'}, \alpha|_{T'})$. Also we write $T(\mathrm{tn})$ to refer to the tasks of tn.

Tasks serve the purpose of unique identifiers because task names (like "move" or "open door") may occur multiple times in the same task network. This leads us to the next definition: two task networks are isomorphic if they describe the same arrangement of task names despite using different identifiers.

We define two task networks $\mathrm{tn} = (T, \prec, \alpha)$ and $\mathrm{tn}' = (T', \prec', \alpha')$ as being *isomorphic*, written $\mathrm{tn} \cong \mathrm{tn}'$, if and only if there exists a bijection $\sigma : T \to T'$, such that for all $t, t' \in T$ it holds that $(t, t') \in \prec$ if and only if $(\sigma(t), \sigma(t')) \in \prec'$ and $\alpha(t) = \alpha'(\sigma(t))$.

Definition 2 (Planning Problem)**.** A *planning problem* is a 6-tuple $\mathcal{P} = (L, C, O, M, c_I, s_I)$ and

- L, a finite set of *proposition symbols*
- C, a finite set of *compound task names*
- O, a finite set of *primitive task names* with $C \cap O = \emptyset$
- $M \subseteq C \times TN_{C \cup O}$, a finite set of *decomposition methods*
- $c_I \in C$, the *initial task name*
- $s_I \in 2^L$, the *initial state*

For each $o \in O$, $(\mathrm{prec}(o), \mathrm{add}(o), \mathrm{del}(o)) \in 2^L \times 2^L \times 2^L$ is called the operator of o and consists of a precondition, an add-, and a delete list. By $\mathrm{tn}_I := (\{t_I\}, \emptyset, \{(t_I, c_I)\})$ we denote a fixed initial task network which consists only of the initial task t_I mapping to the initial task name c_I.

HTN planning formalizations usually feature an initial task network instead of only a single initial task name. We would like to note that our choice does not reduce the class of problems that can be expressed.

Both compound and primitive task names will be used to label the elements of task networks. We refer to tasks that are labeled with elements from C and O as *compound tasks* and *primitive tasks*, respectively. We refer to a task network over O as a *primitive task network*.

As during the rest of this paper there is always exactly one planning problem under consideration, we do not write out "$\mathcal{P} = (L, C, O, M, c_I, s_I)$" each time. Thus, whenever any of the symbols L, C, O, M, c_I, and s_I are used, then they are part of the planning problem $\mathcal{P}$.

After we have defined the basic components of a planning problem, we continue by describing how an initial task network can be transformed into a solution by decomposition and task insertion.

The decomposition of a compound task results in its removal from the task network, followed by an insertion of a copy of the method's task network. The ordering constraints on the removed task are inherited by its replacement tasks (cf. the set $\prec_X$ in the next definition).

Definition 3 (Decomposition)**.** A method $m = (c, \mathrm{tn}_m)$ decomposes a task network $\mathrm{tn}_1 = (T_1, \prec_1, \alpha_1)$ into a new task network tn_2 by replacing task t, written $\mathrm{tn}_1 \xrightarrow[t,m]{} \mathrm{tn}_2$, if and only if $t \in T_1$, $\alpha_1(t) = c$, and there exists a task network $\mathrm{tn}' = (T', \prec', \alpha')$ with $\mathrm{tn}' \cong \mathrm{tn}_m$ and $T' \cap T = \emptyset$, and[3]

$$\begin{aligned} \mathrm{tn}_2 &:= ((T_1 \setminus \{t\}) \cup T', \prec_1 \cup \prec' \cup \prec_X, \alpha_1 \cup \alpha') \text{ with} \\ \prec_X &:= \{(t_1, t_2) \in T_1 \times T' \mid (t_1, t) \in \prec_1\} \cup \\ &\quad\ \{(t_1, t_2) \in T' \times T_1 \mid (t, t_2) \in \prec_1\} \end{aligned}$$

Given a planning problem $\mathcal{P}$, we write $\mathrm{tn}_1 \to_D^* \mathrm{tn}_2$, if tn_2 can be decomposed from tn_1 by an arbitrary number of decompositions using methods from M.

Note that $\to_D^*$ is reflexive and transitive. Also note that the constructed set of ordering constraints $\prec_1 \cup \prec' \cup \prec_X$ is again a strict partial order.

Other formalizations of hybrid planning complement decomposition with the insertion of both primitive and compound tasks, where compound tasks also show preconditions and effects [Kambhampati *et al.*, 1998; Schattenberg and Biundo, 2006]. Since the result of inserting a compound task can be simulated by inserting a decomposed version directly, we only allow the insertion of primitive tasks.

Definition 4 (Task Insertion)**.** Given a task network $\mathrm{tn}_1 = (T_1, \prec_1, \alpha_1)$ and a primitive task name o, then a task network tn_2 can be obtained from tn_1 by insertion of o, if and only if $\mathrm{tn}_2 = (T_1 \cup \{t\}, \prec_1, \alpha_1 \cup \{(t, o)\})$ for some $t \notin T_1$.

[3] For correctness, ordering and labeling of tn_2 must be restricted to $(T_1 \setminus \{t\}) \cup T'$; omitted for better readability.

A task network $\mathrm{tn}_2 = (T_1, \prec_2, \alpha_1)$ can be obtained from tn_1 by insertion of an ordering constraint (t, t'), if and only if $t, t' \in T_1$ and $\prec_2$ is minimal and a strict partial order such that $\prec_1 \cup \{(t, t')\} \subseteq \prec_2$.

Given a planning problem $\mathcal{P}$, we say that tn_2 can be obtained from tn_1 via *task insertion*, written $\mathrm{tn}_1 \rightarrow_I^* \mathrm{tn}_2$, if tn_2 can be obtained from tn_1 by an arbitrary number of insertions of orderings and primitive task names from O.

Note that $\rightarrow_I^*$ is reflexive and transitive.

Definition 5 (Executable Task Network)**.** Given a planning problem $\mathcal{P}$, then a task network $\mathrm{tn} = (T, \prec, \alpha)$ is *executable in state* $s \in 2^L$, if and only if it is primitive and there exists a linearization of its tasks $t_1, \ldots, t_n$ that is compatible with $\prec$ and a sequence of states $s_0, \ldots s_n$ such that $s_0 = s$ and $\mathrm{prec}(\alpha(t_i)) \subseteq s_{i-1}$ and $s_i = (s_{i-1} \setminus \mathrm{del}(\alpha(t_i))) \cup \mathrm{add}(\alpha(t_i))$ for all $1 \leq i \leq n$. We call s_n *the state generated by* tn.

Definition 6 (Solution)**.** A task network tn_S is a solution to a planning problem $\mathcal{P}$, if and only if (1) tn_S is executable in s_I and (2) $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_S$ for tn_S being an *HTN* solution to $\mathcal{P}$ or (2') there exists a task network tn_B such that $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_B \rightarrow_I^* \mathrm{tn}_S$ for tn_S being a *hybrid* solution to $\mathcal{P}$. $\mathrm{Sol}_{\text{HTN}}(\mathcal{P})$ and $\mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ denote the set of all HTN and hybrid solutions of $\mathcal{P}$, respectively.

For the purpose of this paper, we refer to *hybrid solutions* as *solutions* and use the term *HTN solutions* if necessary.

Note that the presented hybrid formalism is capable of capturing ground classical planning. The only missing piece in comparison to classical planning is the goal condition. This can be simulated by having an initial task network that forces an artificial final task, carrying the goal condition in its precondition as the last task in every solution.

3 On the (Un-)Decidability of HTN Planning

When using the HTN solution criterion, we talk about HTN planning, rather than hybrid planning. Although our formalization of HTN planning is inspired by Erol *et al.*'s formalization [1994], it is still a simplification of the latter. As we are going to show the decidability of our hybrid planning formalism, the question arises whether this is due to these simplifications or due to the addition of task insertion. To address this question we reproduce Erol *et al.*'s proof of the undecidability of the plan existence problem for HTN planning [Erol *et al.*, 1996] using *our* formalization, which requires only minor adaptations. In particular, their proof relies on the usage of truth constraints, which are not available in our formalization. And it turns out that they are not necessary for the proof.

We begin by defining the plan existence problem for HTN planning as the set of planning problems that possess an HTN solution, i.e., $\text{PLAN-EXISTENCE}_{\text{HTN}} := \{\mathcal{P} \mid \mathcal{P}$ is a planning problem and $\mathrm{Sol}_{\text{HTN}}(\mathcal{P}) \neq \emptyset\}$.

Theorem 1. *PLAN-EXISTENCE*$_{\text{HTN}}$ *is undecidable*

The proof bases on the fact that one can imitate the production rules of context-free grammars (CFGs) by using decomposition methods. We reduce the undecidable question of whether the languages of two CFGs have a non-empty intersection [Hopcroft *et al.*, 2000, page 407] to the plan existence problem. The idea is to start with a task network that contains the start symbols from both grammars in parallel. These are decomposed into words from the respective grammar's language. The pre- and postconditions of the primitive tasks enforce a final plan in which the symbols of both words are mixed together like the teeth of a zipper. This allows us to ensure that the decomposed task network is executable if and only if the produced words are equal.

Proof. Let G_1, G_2 be two CFGs in Chomsky normal form (each rule has either the form $X \rightarrow YZ$ or $X \rightarrow a$, where X, Y, and Z are non-terminals and a is a terminal) defined over the same set of terminal symbols Σ. For $i \in \{1, 2\}$, we denote by Γ_i the non-terminal symbols of G_i with $S_i \in \Gamma_i$ being the start symbol of G_i. We assume $\Gamma_1 \cap \Gamma_2 = \emptyset$ and define $\Gamma := \Gamma_1 \cup \Gamma_2$. By $L(G_i)$ we denote the language generated by G_i and assume $\varepsilon \notin L(G_1) \cup L(G_2)$. We refer to the grammar rules of G_i by R_i and set $\mathrm{R} := \mathrm{R}_1 \cup \mathrm{R}_2$. We construct $\mathcal{P} = (L, C, O, M, c_I, s_I)$ with $c_I \notin \Gamma$ as follows:

- $L \; := \{\mathit{turn}_{G_1}, \mathit{turn}_{G_2}\} \cup \Sigma$
- $C \; := \Gamma \cup \{c_I\}, O := \{G_i^a \mid a \in \Sigma,\ i \in \{1, 2\}\} \cup \{F\}$
- $M := \{(c_I, \mathrm{tn})\} \cup$
  $\{(X, \mathrm{tn}_{X \rightarrow YZ}) \mid X \rightarrow YZ \in \mathrm{R}\} \cup$
  $\{(X, \mathrm{tn}_{X \rightarrow a}) \mid X \rightarrow a \in \mathrm{R}_i, i \in \{1, 2\}\}$, where
  $$\begin{aligned} \mathrm{tn} &:= (\{s_1, s_2, f\}, \{(s_1, f), (s_2, f)\}, \\ &\qquad \{(s_1, S_1), (s_2, S_2), (f, F)\}) \\ \mathrm{tn}_{X \rightarrow YZ} &:= (\{t, t'\}, \{(t, t')\}, \{(t, Y), (t', Z)\}) \\ \mathrm{tn}_{X \rightarrow a} &:= (\{t\}, \emptyset, \{(t, G_i^a)\}) \end{aligned}$$
- $s_I := \{\mathit{turn}_{G_1}\}$

The operator of $G_1^a \in O$ is defined as $(\{\mathit{turn}_{G_1}\}, \{\mathit{turn}_{G_2}, a\}, \{\mathit{turn}_{G_1}\})$ and that of $G_2^a \in O$ as $(\{\mathit{turn}_{G_2}, a\}, \{\mathit{turn}_{G_1}\}, \{\mathit{turn}_{G_2}, a\})$, respectively. This construction ensures strict turn-taking and a matching of the produced words. The operator of F is defined as $(\{\mathit{turn}_{G_1}\}, \emptyset, \emptyset)$; it ensures that the plan finishes with an operator of the second grammar. By construction, there is a word $\omega \in L(G_1) \cap L(G_2)$ if and only if $\mathcal{P}$ has a solution. □

We have shown that one can express the CFG intersection problem within our HTN formalization, thus proving the undecidability of our planning formalism without insertion. Please note that a solution to the same planning problem using the *hybrid* solution criterion can produce solutions which do *not* correspond to words in the intersection of $L(G_1)$ and $L(G_2)$ due to the the arbitrary insertion of tasks.

4 On the Decidability of HTN Planning with Task Insertion

We now present our proof that yields an upper bound on the length of shortest solutions to a given planning problem. The following paragraph summarizes its main idea.

Suppose there exists a solution tn_S. This solution features a decomposition tn_B of the initial task network. In analogy to the pumping lemma for CFGs [Hopcroft *et al.*, 2000, page 274 ff.], we show that there exists also a "short" decomposition tn'_B. It is obtained by "pumping-down" tn_B, removing all parts that were produced by cycles in the decomposition process. We show that tn'_B can be developed into tn_B by task insertion, which in turn can be developed into tn_S.

Further we provide an upper bound on the number of tasks that must be inserted in order to turn a given task network into a solution, if possible. Applying this result to the short decomposition tn'_B, we can prove the existence of a short solution tn'_S. So in order to decide whether a solution to a planning problem exists, it suffices to check all task networks of a certain size.

4.1 Representing Task Decomposition

This section transfers the idea of a parse tree from formal grammars to the area of task network decompositions, where we call it a *decomposition tree*. It is a representation of how the initial compound task can be decomposed into a primitive task network.

Definition 7 (Decomposition Tree)**.** Given a planning problem $\mathcal{P}$, then a *decomposition tree* $g = (T, E, \prec, \alpha, \beta)$ is a five-tuple with the following properties. (T, E) is a tree with nodes T and directed edges E pointing towards the leafs. There is a strict partial order defined over the nodes, given by $\prec$. The nodes are labeled with task names by $\alpha : T \to C \cup O$. Additionally $\beta : T \to M$ labels inner nodes with methods.

We write $T(g)$ to refer to the tasks of g and $ch(g, t)$ to refer to the direct children of $t \in T(g)$ in g.

The following definition states under which circumstances a decomposition tree encodes a decomposition of the initial compound task. Note that a task network resulting from such a decomposition is *not* necessarily executable.

Definition 8 (Validity of Decomposition Tree)**.** A decomposition tree $g = (T, E, \prec, \alpha, \beta)$ is *valid* with respect to a planning problem $\mathcal{P}$, if and only if the root node of g is labeled with the initial task name c_I and for any inner node t, where $(c, \mathrm{tn}_m) := \beta(t)$, the following holds:

1. $\alpha(t) = c$
2. the task network induced in g by $ch(g, t)$ and tn_m are isomorphic, i.e., $(ch(g, t), \prec|_{ch(g,t)}, \alpha|_{ch(g,t)}) \cong \mathrm{tn}_m$
3. for all $t' \in T$ and all $c' \in ch(g, t)$ it holds that
   (a) if $(t, t') \in \prec$ then $(c', t') \in \prec$
   (b) if $(t', t) \in \prec$ then $(t', c') \in \prec$
4. there are no ordering constraints in $\prec$ other than those demanded by either 2. or 3.

The first criterion ensures the applicability of the methods, the inner nodes are labeled with; the second criterion ensures that the method's task networks are correctly represented within the decomposition tree; and the third criterion ensures the inheritance of ordering constraints as demanded by Definition 3. Please note that every task is uniquely used by g, as we require (T, E) to be a tree.

Definition 9 (Yield of Decomposition Tree)**.** The yield of a decomposition tree $g = (T, E, \prec, \alpha, \beta)$, written $\mathrm{yield}(g)$, is a task network defined as follows. Let $T' \subseteq T$ be the set of all leaf nodes of g. Then, $\mathrm{yield}(g) := (T', \alpha|_{T'}, \prec|_{T'})$.

Proposition 1. *Given a planning problem $\mathcal{P}$, then for any task network $\mathrm{tn} \in TN_{C \cup O}$ the following holds. There exists a valid decomposition tree g with $\mathrm{yield}(g) = \mathrm{tn}$, if and only if $\mathrm{tn}_I \to_D^* \mathrm{tn}$.*

Proof Sketch. For the forward implication, make an induction over the number of inner nodes of g. Note that each inner node corresponds to one method application. As base case, we have the valid decomposition tree without inner nodes, that must consist of only one task that is labeled with c_I and hence its yield is the initial task network tn_I. For the inductive step, fix a valid decomposition tree g with $n + 1$ inner nodes. Let t be an inner node, for which all children $ch(g, t)$ are leafs. Then consider the tree g' which has all children of t removed. Show that g' is valid based on the validity of g. Then show that $\mathrm{yield}(g') \xrightarrow[t,\beta(t)]{} \mathrm{yield}(g)$.

For the backward implication, make an induction over the length of the decomposition sequence. The base case holds due to the reflexivity of $\to_D^*$. In the inductive step, construct the tree by adding newly inserted tasks as children of the replaced task. □

Given a decomposition tree $g = (T, E, \prec, \alpha, \beta)$ and a node $t \in T$, we define the subtree of g induced by t, written $g[t]$, as $g[t] := (T', E', \prec|_{T'}, \alpha|_{T'}, \beta|_{T'})$, where (T', E') is the subtree in (T, E) that is rooted at t. We now define the operation of replacing a subtree by another subtree.

The result of this operation as defined by us is only reasonable for our particular use-case. A general subtree substitution operation would have to create an isomorphic copy of the inserted subtrees, which we have omitted for simplicity.

Definition 10 (Subtree Substitution)**.** Let $g = (T, E, \prec, \alpha, \beta)$ be a decomposition tree and $t_i, t_j \in T$ be two nodes of g. If t_i is the root node of g, then we define the result of the subtree substitution on g that substitutes t_i by t_j, written $g[t_i \leftarrow t_j]$, as $g[t_i \leftarrow t_j] := g[t_j]$; otherwise, $g[t_i \leftarrow t_j] := (T', E', \prec|_{T'}, \alpha|_{T'}, \beta|_{T'})$ with

- $T' := (T \setminus T(g[t_i])) \cup T(g[t_j])$
- $E' := E|_{T'} \cup \{(p, t_j)\}$, where p is the parent node of t_i

The following proposition states that the result of a subtree substitution still describes decompositions if applied under the right circumstances. Also refer to Figure 1 for an illustration of the operation.

Proposition 2. *Given a valid decomposition tree $g = (T, E, \prec, \alpha, \beta)$ with respect to a planning problem $\mathcal{P}$ and two nodes $t_i \in T$, $t_j \in T(g[t_i])$ with $\alpha(t_i) = \alpha(t_j)$, then $g[t_i \leftarrow t_j]$ is also a valid decomposition tree with respect to $\mathcal{P}$.*

Proof Sketch. We have to show that $g' := g[t_i \leftarrow t_j] = (T', E', \prec', \alpha', \beta')$ is still a tree, its root node is labeled with c_I, and that $\prec$ is minimal such that the first three criteria of Definition 8 hold for all inner nodes of g'. For the remainder, let $p \in T$ be such that $(p, t_i) \in E$.

For the special case of the subtree substitution, in which t_i is the root node of g, there is nothing to show, since $g' = g[t_j]$ is clearly a valid decomposition tree with respect to $\mathcal{P}$. Thus, let t_i be different from the root node. It is also easy to see that g' is a tree and its root node is labeled with c_I.

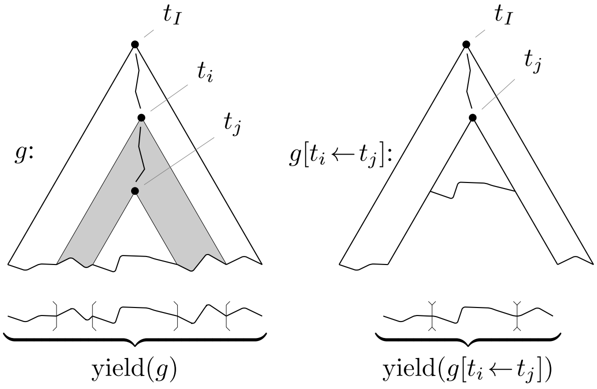


Figure 1: On the left, a decomposition tree g is depicted. The result of the subtree substitution that replaces the subtree below t_i with the subtree below t_j is shown on the right. The gray area inside g corresponds to the tasks that get removed during the substitution. Note how the subtree substitution affects the yield of the tree by removing the contribution of the gray area.

Criterion 1 holds for all inner nodes of g' as both α' and β' are defined on the same nodes and $\alpha' \subseteq \alpha$ and $\beta' \subseteq \beta$.

For criterion 2, we fix some inner node $t \in T'$ and consider the two cases that either $t \neq p$ or $t = p$. The case in which $t \neq p$ is straight forward as the task network induced by $ch(g, t)$ in g is the same as the one induced by $ch(g', t)$ in g'. If $t = p$, the induced task network in g' is still isomorphic to the one in g, because we substituted t_i by t_j and $\alpha(t_i) = \alpha(t_j)$.

To show criterion 3 (considering only 3a, as 3b is analogous), take some ordering constraint (t, t') in $\prec'$. It must also occur in $\prec$ and thus $(c', t') \in \prec$ for all children c' of t. Show $(c', t') \in \prec'$ if $c' \neq t_i$ and $(t_j, t') \in \prec'$, otherwise.

Proving the minimality of $\prec'$ is the most difficult part. We have to show for each $(t_1, t_2) \in \prec'$ that it is required by criterion 2 or 3. Consider the cases that $(t_1, t_2) \in \prec$ is originally produced by criterion 2 (case I) or 3 (case II).

For case I, show that as a consequence (t_1, t_2) is required in $\prec'$ because of criterion 2 with special treatment for $t_1, t_2 \in ch(g', p)$. Intuitively all decompositions in g' have counterparts in g and they produce the same ordering in both trees.

For proving case II (again, we consider only 3a), let $(c', t') := (t_1, t_2)$. We know that there exists a node $t \in T$ with $(t, c') \in E$ and $(t, t') \in \prec$, as this is the antecedent of criterion 3a. Now consider the cases of $t \in T'$ (case IIa) and $t \notin T'$ (case IIb). The proof for case IIa is straight forward. For case IIb, we can conclude that $c' = t_j$ since t_j is the only node in g' that has lost its parent in g. Show that $(p, t') \in \prec'$ and use this as premise to apply criterion 3a in order to show that (c', t') must be in $\prec'$.

□

4.2 Bounding Solution Sizes by Eliminating Cycles

In order to prove our main theorem, which establishes a size limit on the smallest solution of a planning problem, we first need to prove three lemmas. The first one allows us to shorten a decomposition sequence that contains a cycle over compound task names. When doing so, we remain able to recreate via task insertion what has been removed. This result is used by the second lemma to give an upper bound on the size of shortest task networks originated from decomposition that can possibly be developed to a solution. Then the third lemma allows us to find a limit to the number of tasks that have to be inserted in order to turn a given task network into a solution.

The proof of the first lemma uses the idea of the proof of the pumping lemma for CFGs to claim the existence of a shorter version of a given decomposition.

Lemma 1. *Given a planning problem $\mathcal{P}$ and a primitive task network* tn *with valid decomposition tree g, for which there exists a path $t_1, \dots, t_n$ in g with $\alpha(t_i) = \alpha(t_j)$ for some i and j with $i < j \leq n$ (the path contains a cycle). Then it holds that* $\mathrm{tn}_I \rightarrow_D^* \mathrm{yield}(g[t_i \leftarrow t_j]) \rightarrow_I^* \mathrm{tn}$.

Proof. Fix the premises from the lemma. We notice that the conditions for applying Proposition 2 are given. Thus, we can state that $g[t_i \leftarrow t_j]$ is valid. Therefore it generates a task network, name it $\mathrm{tn}' = (T', \prec', \alpha')$, that can be obtained from tn_I by decomposition. It remains to show that $\mathrm{tn}' \rightarrow_I^* \mathrm{tn}$. Since tn' consists of the leafs of $g[t_i \leftarrow t_j]$ and $\mathrm{tn} = (T, \prec, \alpha)$ is induced by the leafs of g, and the substitution only takes away nodes and ordering constraints from g and does not create new leaf nodes, clearly $T' \subseteq T$ and $\prec' \subseteq \prec$ and also $\alpha|_{T'} = \alpha'$. We can thus obtain tn from tn' by adding additional tasks and ordering constraints. □

By using this result repeatedly, we are now able to remove all cycles from a decomposition tree. This makes it possible to formulate an upper bound on the size of the tree and thus of the generated task network.

Lemma 2. *Given a planning problem $\mathcal{P}$, then for every task network* tn *with* $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}$*, there exists a task network* tn' *with* $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}' \rightarrow_I^* \mathrm{tn}$ *and* $|T(\mathrm{tn}')| \leq b^{|C|}$*, where b is the number of tasks inside the largest task network of the methods from M.*

Proof. Fix a planning problem $\mathcal{P}$ and a task network tn with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}$. Fix a decomposition tree g of tn. If g contains a cycle on one of its paths (i.e., two nodes labeled with the same compound task symbol), then Lemma 1 allows us to remove it, obtaining a new decomposition tree g_1 and corresponding task network $\mathrm{tn}_1 = \mathrm{yield}(g_1)$ with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_1 \rightarrow_I^* \mathrm{tn}$.

As long as the new decomposition tree still contains a cycle, we can repeat the procedure. By doing so, we obtain a sequence of decomposition trees $g_1, \dots, g_n$ and corresponding task networks $\mathrm{tn}_1, \dots, \mathrm{tn}_n$. The last decomposition tree g_n does not contain a cycle on any of its paths. The sequence is finite, since every subtree substitution removes at least one task, thus $n \leq |T(g)|$. For the sequence of task networks, it holds that $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_n \rightarrow_I^* \mathrm{tn}_{n-1} \rightarrow_I^* \dots \rightarrow_I^* \mathrm{tn}_1 \rightarrow_I^* \mathrm{tn}$ and by the transitivity of task insertion we get $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_n \rightarrow_I^* \mathrm{tn}$. We choose tn_n as the tn' from the lemma. It remains to show the size bound $|T(\mathrm{tn}_n)| \leq b^{|C|}$.

The size of tn_n is limited because its decomposition tree g_n contains no cycle on any of its paths and is thus bounded in depth by $|C|$. Taking the maximal branching factor b of g, we get an upper bound of $b^{|C|}$ of the number of leaf nodes of tn_n and thus on the number of tasks of $\mathrm{tn}_n = \mathrm{yield}(g)$. □

While the first two lemmas have somehow tamed the decomposition aspect of the solution criteria, we still need to care for executability. The idea is to take the tasks that have been introduced by decomposition and connect them in an executable way by using task insertion. Thus we obtain one classical planning problem for each task inside the decomposed task network.

Proposition 3. *Let $\mathcal{P}$ be a planning problem and $s, s' \in 2^L$. If there exists a task network* tn *which is executable in s and generates s', then there is also a task network* tn' *which is executable in s, generates s', and $|T(\mathrm{tn}')| \leq 2^{|L|}$.*

Proof. As the size of the state space induced by $\mathcal{P}$ is $2^{|L|}$, each task network longer than this value must traverse at least one state twice and hence must contain a cycle in the task sequence which can simply be omitted. □

Lemma 3. *Given a planning problem $\mathcal{P}$ and a task network* tn*, that can be developed into a solution $\mathrm{tn}_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ via task insertion $\mathrm{tn} \rightarrow_I^* \mathrm{tn}_S$, then there exists a solution $\mathrm{tn}'_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ with $|T(\mathrm{tn}'_S)| \leq |T(\mathrm{tn})|\,(2^{|L|}+1)$.*

Proof. Since tn_S is a solution to $\mathcal{P}$, there exists an executable linearization Lin_S of its tasks. Let $\mathrm{Lin} := t_1, t_2, \ldots, t_n$ be the tasks of tn ordered as they appear inside Lin_S. Let s_i and s'_i be the states before and after the execution of t_i when applying Lin_S to the initial state s_I.

In order to develop Lin into an executable task sequence, we need to find the $n-1$ task networks tn_i that transform s'_i into s_{i+1} for $0 < i < n-1$ and the task network tn_0 that transforms s_I into s_1. Proposition 3 tells us that we can find such task networks containing at most $2^{|L|}$ many tasks each, if these problems are solvable at all. And we can use Lin_S as a witness, that they are solvable.

Putting it all together, we can construct a solution task network illustrated by the sequence $\mathrm{tn}_0 t_1 \mathrm{tn}_1 t_2 \ldots \mathrm{tn}_{n-1} t_n$. This task network contains at most $n2^{|L|} + n$ many tasks and can be constructed from tn via task insertion. □

Theorem 2. *Given a planning problem $\mathcal{P}$ with $\mathrm{Sol}_{\text{HYBRID}}(\mathcal{P}) \neq \emptyset$, then there exists a solution $\mathrm{tn}^*_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$ with $|T(\mathrm{tn}^*_S)| \leq b^{|C|}(2^{|L|}+1)$, where b is the number of tasks inside the largest task network of the methods from M.*

Proof. A graphical presentation of this proof is given in Figure 2. Fix a solution $\mathrm{tn}_S \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})$. There exists a task network tn_B with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}_B$ and $\mathrm{tn}_B \rightarrow_I^* \mathrm{tn}_S$ because of the solution criteria. This task network marks the boundary between the decomposition part and the insertion part of the planning process. By Lemma 2 there exists a (not necessarily different) task network tn'_B with $\mathrm{tn}_I \rightarrow_D^* \mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_B$ and $|T(\mathrm{tn}'_B)| \leq b^{|C|}$. Because of transitivity of task insertion, from $\mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_B \rightarrow_I^* \mathrm{tn}_S$, it follows that $\mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_S$.

Using Lemma 3, from $\mathrm{tn}'_B \rightarrow_I^* \mathrm{tn}_S$, it follows that there exists a solution tn^*_S with $|T(\mathrm{tn}^*_S)| \leq |T(\mathrm{tn}'_B)|\,(2^{|L|}+1)$ and thus $|T(\mathrm{tn}^*_S)| \leq b^{|C|}(2^{|L|}+1)$. □

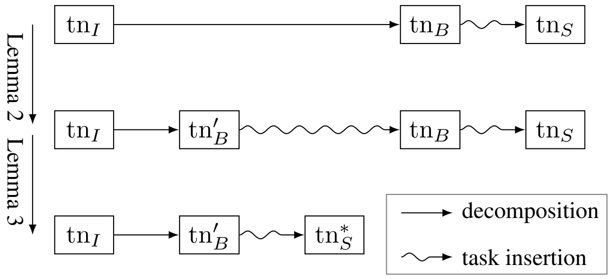


Figure 2: This figure shows how the proof of Theorem 2 constructs a small solution tn^*_S from a large solution tn_S. The task network tn_B marks the boundary between task decomposition and task insertion. The length of the arrows corresponds to the number of applied modifications, and thus correlates to the size of the resulting task network.

4.3 Complexity Results

In this section we will put the previous results to use in order to obtain an upper complexity bound of **EXPSPACE** for the plan existence problem for propositional, hybrid planning.

First we are going to define two decision problems. As usual, these are represented by subsets of words over a given alphabet Σ. We do not deal with the problem of encoding the syntactical structures, such as task networks or problems, into words over Σ. It is assumed that this process and the rejection of malformed inputs is feasible and does not add to the complexity of the problem. We denote the length of the description of an object X in Σ by $|X|_\Sigma$. As a remark about task networks, note that the description of a task network tn is bounded by $O(|T(\mathrm{tn})|^2)$ because of the possible number of ordering constraints.

We define the decision problem $\text{SOLUTION}_{\text{HYBRID}}$ as all tuples of a planning problem and a corresponding hybrid solution, i.e., $\text{SOLUTION}_{\text{HYBRID}} := \{(\mathcal{P}, \mathrm{tn}) \mid \mathcal{P} \text{ is a planning problem and } \mathrm{tn} \in \mathrm{Sol}_{\text{HYBRID}}(\mathcal{P})\}$. Then we can state the following proposition.

Proposition 4. *SOLUTION*$_{\text{HYBRID}} \in$ **PSPACE**.

Proof. Take as input the tuple $(\mathcal{P}, \mathrm{tn})$. Since we demand that a task network may not be empty, the application of a decomposition to a task network never decreases its number of tasks $|T(\mathrm{tn})|$. This means that during a decomposition $\mathrm{tn}_1 \rightarrow_D^* \mathrm{tn}_2$, the intermediate task networks never exceed $\mathrm{pol}(|\mathrm{tn}_2|_\Sigma)$. Obviously the same holds for a sequence of insertions. We can thus state the following nondeterministic algorithm, that runs in polynomial space in the size of $|(\mathcal{P}, \mathrm{tn})|_\Sigma$ and thus in **PSPACE**. (1) Check if tn is executable in s_I; reject if not. (2) The following works in-place: non-deterministically decompose tn_I into a primitive task network tn' with size of at most $\mathrm{pol}(|\mathrm{tn}|_\Sigma)$; then, non-deterministically insert tasks and ordering constraints not violating this size bound; check if tn was produced; accept if so, otherwise reject. □

We define the plan existence problem for hybrid planning as the set of planning problems that possess a hybrid solution, i.e., PLAN-EXISTENCE$_{\text{HYBRID}}$:= $\{\mathcal{P} \mid \mathcal{P}$ is a planning problem and $\text{Sol}_{\text{HYBRID}}(\mathcal{P}) \neq \emptyset\}$.

Corollary 1. *PLAN-EXISTENCE*$_{\text{HYBRID}} \in \mathbf{EXPSPACE}$.

Proof. Let the input be $\mathcal{P}$ and let $n = |\mathcal{P}|_\Sigma$ be the length of this input. Theorem 2 states that there exists a solution tn_S^* with $|T(\text{tn}_S^*)| \leq b^{|C|}(2^{|L|} + 1)$, if and only if $\mathcal{P}$ has a solution at all. From the bound on the number of tasks and by substituting b, $|C|$, and $|L|$ by n, we can estimate $|\text{tn}_S^*|_\Sigma \leq \text{pol}(n^n(2^n + 1))$. Thus, $|\text{tn}_S^*|_\Sigma \leq 2^{\text{pol}(n)}$ and the following non-deterministic algorithm runs in exponential space. (1) Non-deterministically guess a task network tn with $|T(\text{tn})| \leq b^{|C|}(2^{|L|} + 1)$. (2) Check whether $(\mathcal{P}, \text{tn}) \in$ SOLUTION$_{\text{HYBRID}}$ using only space polynomial in $|(\mathcal{P}, \text{tn})|_\Sigma \leq n + 2^{\text{pol}(n)}$; accept if the check returns true, otherwise reject. The given algorithm decides PLAN-EXISTENCE$_{\text{HYBRID}}$ using only space exponential in the input length. □

And as a direct consequence, we conclude:

Corollary 2. *PLAN-EXISTENCE*$_{\text{HYBRID}}$ *is decidable.* □

5 Conclusion and Discussion

We have formalized a simplified, propositional version of HTN planning with task insertion (or hybrid planning) that allows for the insertion of tasks and ordering constraints. We have established $\mathbf{EXPSPACE}$ as an upper complexity bound of the corresponding plan existence problem. We have also shown that plan existence is undecidable given our formalization *without insertion*. We conclude that the possibility to insert tasks as an addition to task decomposition greatly reduces the computational complexity of HTN planning.

A direct application of the result to the established HTN formalism by Erol *et al.* [1994] might not be possible, since undecidability could have been "reintroduced" by a feature that is not captured by us, like the formulation of truth constraints. But even then we would have eliminated at least one source of undecidability.

Schattenberg and Biundo [2006] solve hybrid planning problems close to the ones defined in this paper. However, the initial task network and the decomposition methods may contain causal links, thereby further restricting the set of possible solutions. Thus, it is not clear, whether our decidability result also applies to their formalization.

However, our result seems to apply to the problems solved by the `Duet` planning system [Gerevini *et al.*, 2008]. It processes problems as defined in this paper, but uses a lifted state representation and features preconditions for decomposition methods. `Duet` could be extended to prune plans that exceed a certain size and achieve guaranteed termination without sacrificing completeness. Note that the complexity result is not directly transferable, since the step from a propositional state representation to a lifted one will most likely result in an exponential increase in complexity.

Since the focus of the paper lies on showing decidability, we did not attempt to provide tight complexity bounds for the plan existence problem. We have shown $\mathbf{EXPSPACE}$ membership and can state $\mathbf{PSPACE{-}hard}$ as a trivial lower bound. This is the case, because our formalization captures ground, classical planning with negative effects and operators given in the input, which has been proved to be $\mathbf{PSPACE{-}complete}$.

Acknowledgements

We want to thank our reviewers for their insightful questions and suggestions that helped us improving this paper.

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

[Erol *et al.*, 1994] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994)*, pages 249–254, 1994.

[Erol *et al.*, 1995] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76:75–88, 1995.

[Erol *et al.*, 1996] Kutluhan Erol, James Hendler, and Dana S. Nau. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93, 1996.

[Estlin *et al.*, 1997] Tara A. Estlin, Steve A. Chien, and Xuemei Wang. An argument for a hybrid HTN/operator-based approach to planning. In *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, pages 182–194, 1997.

[Gerevini *et al.*, 2008] Alfonso Gerevini, Ugur Kuter, Dana S. Nau, Alessandro Saetti, and Nathaniel Waisbrot. Combining domain-independent planning and HTN planning: The duet planner. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pages 573–577. IOS Press, 2008.

[Hopcroft *et al.*, 2000] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, volume 3. Addison-Wesley Reading, MA, second edition, 2000.

[Kambhampati *et al.*, 1998] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. Hybrid planning for partially hierarchical domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*, pages 882–888. AAAI Press, 1998.

[Schattenberg and Biundo, 2006] Bernd Schattenberg and Susanne Biundo. A unifying framework for hybrid planning and scheduling. In *Advances in Artificial Intelligence, Proceedings of the 29th German Conference on Artificial Intelligence (KI 2006)*, pages 361–373. Springer, 2006.

The following pages show the publication:

P. Bercher, G. Behnke, D. Höller, and S. Biundo. “An Admissible HTN Planning Heuristic”. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*. AAAI Press, 2017, pp. 480–488. DOI: 10.24963/ijcai.2017/68

Reprinted with kind permission of AAAI Press.

The included PDF is a revised version. Modifications are not explicitly mentioned, since there were only minor corrections or improvements.

An Admissible HTN Planning Heuristic

Pascal Bercher, Gregor Behnke, Daniel Höller, Susanne Biundo
Institute of Artificial Intelligence, Ulm University, Ulm, Germany
{pascal.bercher, gregor.behnke, daniel.hoeller, susanne.biundo}@uni-ulm.de

Abstract

Hierarchical task network (HTN) planning is well-known for being an efficient planning approach. This is mainly due to the success of the HTN planning system SHOP2. However, its performance depends on hand-designed search control knowledge. At the time being, there are only very few domain-independent heuristics, which are designed for differing hierarchical planning formalisms. Here, we propose an admissible heuristic for standard HTN planning, which allows to find optimal solutions heuristically. It bases upon the so-called task decomposition graph (TDG), a data structure reflecting reachable parts of the task hierarchy. We show (both in theory and empirically) that rebuilding it during planning can improve heuristic accuracy thereby decreasing the explored search space. The evaluation further studies the heuristic both in terms of plan quality and coverage.

1 Introduction

In contrast to classical planning, the goal in hierarchical planning is not to find a plan that satisfies a goal formula that holds in the state produced by executing the plan, but to find an executable refinement of an initial partial plan (a partially ordered set of primitive and/or abstract tasks). This initial plan is thus the "goal" – the set of tasks one wants to have achieved. Partial plans consist of two kinds of tasks: primitive and abstract ones. As it is the case in classical planning, only primitive ground tasks (so-called actions) are directly executable in a state, while abstract tasks need to be refined into more primitive ones until an executable primitive (solution) plan is obtained.

The motivation for choosing a hierarchical problem class as underlying framework is manifold, since the task hierarchy can be exploited in several ways. In case a user is integrated into the plan generation process, a hierarchical planning approach seems more natural, since it shows many similarities with the way in which humans solve their problems [Byrne, 1977; Fox, 1997; Marthi *et al.*, 2008]. For the same reason, plan explanations ("Why do I have to perform this action?") can make use of the hand-designed hierarchy to give intuitive explanations [Seegebarth *et al.*, 2012; Bercher *et al.*, 2014a]. The task hierarchy also allows to capture constraints restricting the set of plans that are regarded solutions, which cannot be expressed without the hierarchy [Höller *et al.*, 2014; 2016] (if no other expressive constructs such as functions are available). This higher expressivity of hierarchical planning problems comes at the cost of higher computational complexities. That is, deciding whether there is a solution is generally harder than in non-hierarchical planning – the precise complexity depends on structural properties of the task hierarchy and is at most undecidable [Erol *et al.*, 1994b; Alford *et al.*, 2015a; Bercher *et al.*, 2016] – and even deciding whether a partial plan is a solution is harder than in non-hierarchical planning [Behnke *et al.*, 2015; Bercher *et al.*, 2016]. Contrary to these seemingly negative results, the hierarchy can also be exploited to reduce the time required to find a solution. That is, instead of (or: in addition to) exploiting the hierarchy to pose additional constraints on solutions, it can be used to encode search control as exploited by the well-known SHOP2 system [Nau *et al.*, 2003].

Like in classical, non-hierarchical, planning, in some application domains one wants to find *optimal* solutions or solutions of a certain quality. Particularly in real-world applications this is of crucial importance, as plan quality often translates to costs in the real world, e.g., in terms of money, time, or resource consumption. Several hierarchical planning approaches have addressed the issue of finding optimal solutions [Lotem *et al.*, 1999; Marthi *et al.*, 2008; Sohrabi *et al.*, 2009; Shivashankar *et al.*, 2016; 2017]. To find such solutions, it suffices to use a suitable algorithm in combination with an admissible heuristic. We propose such an admissible heuristic for standard HTN planning.

Our heuristic is based upon the so-called task decomposition graph (TDG) – a representation of the AND/OR structure underlying the task hierarchy [Elkawkagy *et al.*, 2012]. It exploits the TDG to find a least-cost set of primitive actions into which the given abstract tasks can be decomposed. We show after which plan modifications during search rebuilding the TDG might lead to improved heuristic estimates. In the empirical evaluation, we show that the heuristic performs slightly worse than inadmissible ones when using A*, but finds plans of better quality. With Greedy-A*, it becomes one of the best configurations. The TDG-recomputation reduces the search space in almost every problem instance, but also increases the runtime for several instances.

2 Related Work

Independent of the purpose that hierarchical planning is deployed for, it is always desirable to solve the respective problems as fast as possible. This is especially important if the hierarchy is used for "physics" instead of "advice" (i.e., if the domain is modeled in a hierarchical manner, but it does not reflect search guidance). Several approaches were proposed to find solutions fast. Some of these approaches pursue the same basic approach than we do in this paper: using a standard search procedure (such as progression search) in combination with a domain-independent heuristic. Other approaches do not (only) rely on domain-independent heuristics, but instead the information encoded in the task hierarchy is exploited by the algorithm itself.

Marthi *et al.* [2008] propose several algorithms for their *angelic hierarchical planning* semantics. They are concerned with generating provably optimal "high-level plans" that may still contain abstract tasks. In their framework, abstract tasks are associated with preconditions and effects, the latter describing the set of states reachable by some refinement of the respective task. They further assume a totally ordered problem (making the problem decidable, because it prevents intertwining plans [Erol *et al.*, 1994b; Alford *et al.*, 2015a]). To find optimal plans, they deploy A* that exploits optimistic and pessimistic estimates of abstract tasks, which estimate the cost of their reachable refinements. They note: *"we will assume that the descriptions are given along with the hierarchy. However, we note that it is theoretically possible to derive them automatically from the structure of the hierarchy."* We believe that our admissible heuristic values can serve as their optimistic estimates that they rely on. Shivashankar *et al.* [2016; 2017] developed an admissible landmark-based heuristic for finding optimal solutions in *Hierarchical Goal Network (HGN) planning*, a formalism closely related to standard HTN planning (the relationship is investigated in detail by Alford *et al.* [2016b]). In standard HTN planning, decomposition methods specify into which task networks (a partially ordered set of tasks) an abstract task may be decomposed. In HGN planning, methods specify into which goal networks (a partially ordered multiset of goals) a goal may be decomposed. The proposed heuristic exploits the close relationship between goal networks and the (partially ordered) landmarks of a problem. Sohrabi *et al.* [2009] are concerned with finding high-quality plans for *HTN planning with preferences*. They propose a branch-and-bound algorithm that first finds some solution quickly (guided by inadmissible heuristics) and then tries to find better solutions by pruning plans that will lead to solutions of equal or worse quality (based on an admissible heuristic). Most heuristics are specific to both progression search (as they exploit that the current state is changing) and to their approach how the preferences are compiled away. One of their heuristics, lookahead metric function, is closer related to ours as it is an estimate of the metric of the best successor to the current partial plan. It does so by first calculating all refinements up to a certain depth. For each of them, a single primitive partial plan is computed based on Depth First search. The best metric serves as heuristic. The *MME heuristic* for hybrid planning [Bercher *et al.*, 2014b] is closely related to the one proposed here. We will detail the precise relationship later in the paper.

The planning system *Duet* [Gerevini *et al.*, 2008] combines the HTN planner SHOP2 [Nau *et al.*, 2003] with LPG [Gerevini *et al.*, 2003], a stochastic search planner for non-hierarchical problems. Duet is not directly concerned with calculating heuristics for partial plans, but it shows how hierarchical problems can be solved efficiently without relying on hand-coded search control. It is also notable that Duet is not concerned with solving standard HTN problems, since it allows to insert actions into partial plans (for achieving goals) that do not stem from task decomposition as allowed in TIHTN planning (HTN planning with task insertion [Geier and Bercher, 2011; Alford *et al.*, 2015b]). The planner *GraphHTN* [Lotem *et al.*, 1999] solves standard HTN problems via combining the so-called *planning tree* with the planning graph. The planning tree is an AND/OR tree that represents all decompositions of the initial partial plan up to a certain depth. We rely on a similar graph representation. GraphHTN works by extending both the planning graph and the planning tree until a solution is found. It guarantees to find a solution with shortest makespan.

3 Problem Formalization

We rely on a problem formalization that extends standard *HTN planning* [Erol *et al.*, 1994b; Geier and Bercher, 2011] with concepts known from *partial order causal link (POCL) planning* [McAllester and Rosenblitt, 1991]. In accordance to previous work, we refer to the respective framework as *hybrid planning* [Biundo and Schattenberg, 2001; Bercher *et al.*, 2016]. Hybrid models extend standard HTN models in two directions: First, abstract tasks syntactically look like primitive ones, i.e., they also have preconditions and effects. That way, the planning model can be restricted to methods that adhere the semantics intended by the modeler [Bercher *et al.*, 2016]. Second, the model's methods can contain causal links that annotate which (abstract or primitive) task's precondition has been achieved by which other task of that method.

We want to emphasize that the the proposed heuristic is neither inherently making use of the preconditions and effects of abstract tasks nor of the causal links, both of which distinguish hybrid models from standard HTN models. As a consequence, the proposed heuristic can cope both with hybrid and with HTN models – i.e., it can be regarded both an HTN as well as a hybrid planning heuristic.

In hybrid planning, both primitive and abstract tasks are 3-tuples $\langle t(\bar{\tau}), \mathit{pre}(\bar{\tau}), \mathit{eff}(\bar{\tau})\rangle$ consisting of a parametrized name $t(\bar{\tau})$, a precondition $\mathit{pre}(\bar{\tau})$, and effects $\mathit{eff}(\bar{\tau})$ – the latter two are conjunctions of literals and depend on the task's parameter variables $\bar{\tau}$. We will often refer to a task by just mentioning its name $t(\bar{\tau})$. Partial plans are partially ordered sets of primitive and/or abstract tasks. A partial plan P is given by a tuple $\langle PS, \prec, CL, VC\rangle$ consisting of its plan steps PS, ordering constraints $\prec$, causal links CL, and variable constraints VC. A plan step $l\!:\!t(\bar{\tau}) \in PS$ is a uniquely labeled task. Labeling is required because P maintains a *partial* order of its plan steps, so multiple occurrences of the same task are differentiated relying on the labels. The set $\prec$ is a strict partial order on PS. A causal link $ps \rightarrow_{\varphi} ps' \in CL$ represents that the precondition φ of plan step ps' is achieved by the plan step ps. The set VC is defined over the variables $\bar{\tau}$ of the tasks of PS. It can co- or non-co-designate variables with each other or with constants. With $\mathit{Ground}_{VC}(P)$ we denote the set of all groundings of P under consideration of the additional variable constraints VC.

Whereas primitive tasks have the same semantics as in classical planning, abstract tasks are abstractions of several primitive or abstract tasks. They can thus be regarded representations of high-level activities that need to be refined into more specific courses of action. This is accomplished by a set of so-called *(decomposition) methods*. A method is a tuple $m = (t(\bar{\tau}), P_m, VC_m)$ that maps an abstract task to its pre-defined partial plan P_m. VC_m denotes a set of variable constraints that relates the variables $\bar{\tau}$ of $t(\bar{\tau})$ with the variables of the partial plan P_m. In our formalism, methods are not associated with preconditions as it is the case in other hierarchical planning formalisms (as exploited, e.g., by SHOP2). That is, given a partial plan P contains a plan step $l : t(\bar{\tau}') \in PS$ and $\bar{\tau}$ and $\bar{\tau}'$ can be unified, then m is applicable to P. Applying m to P results in a successor plan P' in which $t(\bar{\tau})$ has been removed and replaced by P_m with the according variable constraints. Any of the causal links involving a precondition or effect of the decomposed abstract task is inherited down to suitable tasks in P_m. Adhering to legal methods ensures that this is always possible [Bercher *et al.*, 2016].

A hybrid planning domain $\mathcal{D} = \langle T_p, T_a, M \rangle$ contains the primitive and abstract tasks T_p and T_a, respectively, and a set of methods M. A hybrid planning problem $\mathcal{P} = \langle \mathcal{D}, P_{init}, C \rangle$ is then given by a domain $\mathcal{D}$, an initial partial plan P_{init}, and a set of constants C. The problem's initial state and, if given, its goal description, are encoded in P_{init} as additional actions as done in POCL planning.

A partial plan P is a solution (plan) to a hybrid planning problem, if and only if the following two criteria hold:

- P is a refinement of P_{init} with respect to the decomposition of abstract tasks and the insertion of ordering constraints, variable constraints, and causal links.
- P is executable in the initial state in the sense of the standard POCL solution criteria. That is, all tasks are primitive and ground, for each precondition φ of some plan step ps' there is a causal link $ps \rightarrow_{\varphi} ps'$ from a plan step ps in P, and there are no causal threats. A plan step ps'' threatens a causal link $ps \rightarrow_{\varphi} ps'$ if and only if it has some effect $\neg\psi$, such that ψ and φ can be unified and ps'' could be ordered between ps and ps'.

Note that we do not allow the insertion of tasks (cf. first solution criterion), except via decomposition of abstract tasks, in order to stick to the standard HTN solution criteria [Erol *et al.*, 1994b; Alford *et al.*, 2015a] as opposed to a relaxation thereof, called HTN planning with task insertion, TIHTN planning [Geier and Bercher, 2011; Alford *et al.*, 2015b].

4 On the Task Decomposition Graph

The so-called *task decomposition graph (TDG)* [Elkawkagy *et al.*, 2012] represents the AND/OR structure of the task hierarchy. Since it is a canonical representation of hierarchical problems, similar structures are used for various purposes (e.g., as the basis for HTN planning systems [Lotem *et al.*, 1999]). We use it to ground the domain model and to estimate the remaining effort of turning a given abstract task into a primitive plan. We first give a formal definition of TDGs, which is equivalent to the one given by Elkawkagy *et al.* [2012], but simplified and therefore more intuitive.

Definition 1 (Task Decomposition Graph (TDG))**.** *Let $\mathcal{P} = \langle \mathcal{D}, P_{init}, C \rangle$ be a hybrid planning problem with domain $\mathcal{D} = \langle T_p, T_a, M \rangle$. Without loss of generality, we assume that P_{init} contains just a single ground abstract task* TOP *for which there is exactly one method in M.*[1]

The bipartite graph $\mathcal{G} = \langle V_T, V_M, E_{T\rightarrow M}, E_{M\rightarrow T} \rangle$, consisting of a set of task vertices V_T, method vertices V_M, and edges $E_{T\rightarrow M}$ and $E_{M\rightarrow T}$ is called the TDG of $\mathcal{P}$ if it holds:

1. ***base case** (task vertex for the given task)*
   TOP $\in V_T$*, the TDG's root.*
2. ***method vertices** (derived from task vertices)*
   Let $v_t \in V_T$ with $v_t = t(\bar{c})$ and $(t(\bar{\tau}), P_m, VC_m) \in M$. Then, for all $v_m \in \mathit{Ground}_{VC_m \cup \{\bar{\tau}=\bar{c}\}}(P_m)$ holds:
   • $v_m \in V_M$ • $(v_t, v_m) \in E_{T\rightarrow M}$.
3. ***task vertices** (derived from method vertices)*
   Let $v_m \in V_M$ with $v_m = \langle PS, \prec, CL, VC \rangle$. Then, for all plan steps $l : t(\bar{c}) \in PS$ with $v_t = t(\bar{c})$, holds:
   • $v_t \in V_T$ • $(v_m, v_t) \in E_{M\rightarrow T}$.
4. ***tightness***
   $\mathcal{G}$ is minimal, such that 1. to 3. hold.

The definition works inductively by first requiring that the problem's initial ground task TOP[1] is part of the TDG as its root. The second criterion, one of the inductive steps, requires that for all methods that are applicable to any task vertex of the TDG, their respective ground partial plans are method vertices of the TDG. The third criterion, the second inductive step, requires any task in any of the TDG's method vertices to be a task vertex of the TDG. Finally, the last criterion ensures minimality of the graph, so that no vertexes or edges are in the TDG other than the ones demanded by the previous criteria. Graphical illustrations of example TDGs are provided by Lotem *et al.* [1999], Elkawkagy *et al.* [2012], and in Fig. 1.

Due to the undecidability of HTN planning [Erol *et al.*, 1994b; Geier and Bercher, 2011] and hybrid planning [Bercher *et al.*, 2016], there cannot always be a limit on the number of method applications to find a solution. However, since the TDG contains each decomposition method and task at most once, the TDG is always finite.

Def.1 incorporates *all* tasks that can be reached via decomposing the initial partial plan. Instead, we deploy the technique by Elkawkagy *et al.* [2010] that removes parts of the TDG which are unreachable when considering a delete-relaxed reachability analysis of the primitive tasks.

[1] If the problem specifies an initial partial plan P_{init} we can obtain the required form by adding a new artificial (parameter-free) abstract task TOP that decomposes exactly into P_{init}.

5 An Admissible Heuristic Based on the TDG

In case TDG-recomputation is not enabled, the proposed heuristic is a pre-processing heuristic, since it calculates the TDG only once before planning and assigns cost estimates to each of its vertices that also do not change. During search, these values are retrieved for a given search node. We assume that we have given a TDG and show how these cost estimates are calculated. To ensure termination, we assume that every primitive ground task $t(\bar{c})$ has a non-negative action cost $\mathit{cost}(t(\bar{c})) \in \mathbb{R}^+$ and that the TDG contains only abstract tasks that can be refined into a set of primitive tasks. Abstract tasks that do not fulfill the latter property can easily be identified in polynomial time by relying on a bottom-up reachability analysis (proof of Thm. 3.1 by Alford *et al.* [2014]).

We estimate the effort of refining an abstract task vertex by minimizing over the estimated effort of its method vertices. Analogously, the effort of refining a method vertex can be estimated by summing over the estimates of the tasks it contains. We do this for each task in the TDG. The semantics behind this calculation can be regarded as the cost of the least-expensive set of primitive tasks into which a given abstract task can be refined. This implies that these tasks must not necessarily form an executable plan, but they can all be made applicable using delete-relaxed primitive tasks, as they would not be in the TDG otherwise.

Definition 2 (TDG Cost Estimates)**.**
Let $\langle V_T, V_M, E_{T\to M}, E_{M\to T}\rangle$ be a TDG.

$$h_T(v_t) := \begin{cases} \mathit{cost}(v_t) & \textit{if } v_t \textit{ is primitive} \\ \min\limits_{(v_t,v_m)\in E_{T\to M}} h_M(v_m) & \textit{else} \end{cases} \qquad (1)$$

For a method vertex $v_m = \langle \mathit{PS}, \prec, \mathit{CL}, \mathit{VC}\rangle$, we set:

$$h_M(v_m) := \sum_{(v_m,v_t)\in E_{M\to T}} h_T(v_t) \qquad (2)$$

We want to emphasize that each cost estimate of the TDG's vertices is finite even though the TDG itself might be cyclic. Intuitively spoken, it can never be optimal (or required) to run into a cycle for the sake of minimizing the estimates.

We use the following algorithm to compute the cost estimates. First, we identify all strongly connected components (SCCs) of the TDG, which is possible in polynomial time. All primitive tasks form an SCC on their own, since they do not have outgoing edges. They can be assigned their cost value according to h_T. We then sort all SCCs topologically, which is possible since their dependencies form a directed acyclic graph. We then process them in their topological (possibly partial) order, starting with the SCCs containing primitive tasks. For the remaining SCCs, we use an iterate-until-fixpoint procedure. We start by initializing the estimates for all vertices in an SCC with ∞. Then we iterate over these vertices and use the formulae for h_T and h_M to update the respective estimates. This iteration is repeated until no value in the SCC has changed, i.e., until the correct estimates according to Def. 2 have been computed. This fixpoint is reached after a polynomial number of steps, because in each iteration at least one vertex in the SCC is assigned its final value, which can be proven as follows. If the SCC contains one vertex, the claim is trivial. Otherwise, there are one or more vertices in it with outgoing edges. Consider a *method vertex* in an SCC: since it has been included, all other vertices are reachable from this one, i.e., at least one of its edges points to a vertex in the SCC. Thus, one of the outgoing edges of the SCC must originate from a task vertex (otherwise there were an infinite recursion and the SCC would have been pruned). Since the costs of a method vertex are additive with respect to its children, the method vertices will be at least as costly as the cheapest task vertex in the SCC. One of the outgoing edges of the SCC originating from a *task vertex* v_t will lead to a method vertex with minimal cost estimate c. As noted, every recursion must leave the SCC via a task vertex, and we have picked the cheapest task vertex, therefore this is the vertex with minimal costs (in the fixpoint). Since all estimates in the SCC must be greater (or equal) to c, the vertex v_t gets assigned c as its fixpoint cost in the first round of iterations. Since we have found the edge that determines the value of v_t, all other outgoing edges can be ignored from now on. Ignoring these edges, v_t falls out of the SCC. As such, the size of the SCC would decrease and, by induction, n iterations suffice to reach the fixpoint for an SCC with n vertices.

Def. 2 and the procedure above further forms an improvement of the so-called *minimal modification effort (MME)* heuristic, which is designed for hybrid planning systems that rely on POCL search techniques [Bercher *et al.*, 2014b]. MME is also based on the idea to exploit a TDG via minimizing over different methods and summing within the same method. But instead of action costs, it estimates the number of required decompositions and causal link insertions a hybrid planner needs to perform: in (1), primitive tasks are estimated by the number of their preconditions and abstract tasks by the minimum as given here plus 1 for performing a decomposition. In (2), we subtract $|CL|$ from the given sum to account for causal links that are already in a partial plan. But in contrast to Def. 2, MME relies on a visited list of abstract tasks that are taken into account when calculating the TDG estimates. This list ensures termination in the presence of cycles (cf. Def. 3 by Bercher *et al.* [2014b]), but it also produces smaller and less accurate estimates in these cases. We give the respective improved version of the MME heuristic later in this section.

During search, given a current partial plan and one of its abstract plan steps $l : t(\bar{\tau})$, we retrieve the estimate of one of its compatible groundings in the TDG, $comp(t(\bar{\tau}))$. When using the TDG cost estimates given in Def. 2, the resulting heuristic is called *cost-aware TDG heuristic*, *TDG$_c$*.

Definition 3 (TDG$_c$ Heuristic)**.**
Let $P = \langle \mathit{PS}, \prec, \mathit{CL}, \mathit{VC}\rangle$ be a partial plan. Then,

$$h_{\mathit{TDG}_c}(P) := \sum_{\substack{l:t(\bar{\tau})\in \mathit{PS} \\ t(\bar{\tau})\ \mathit{abstract}}} \left(\min_{v_t \in comp(t(\bar{\tau}))} h_T(v_t) \right)$$

TDG$_c$ basically relies upon the same mechanics as the definition of the method vertex estimate in Def. 2, since both inputs are partial plans. The heuristic for the costs of any missing actions can be estimated by summing over the heuristic estimates of its abstract tasks (which were pre-calculated, cf. Def. 2). Since these tasks might still be lifted in the given partial plan, we minimize over the possible groundings.

Since Def. 2 minimizes for each abstract task over its available methods, and again Def. 3 minimizes over all possible groundings, the resulting heuristic is clearly admissible.

Prop 1. *The cost-aware TDG heuristic h_{TDG_c} is admissible with respect to action costs.*

Note that removing unreachable parts of the TDG [Elkawkagy *et al.*, 2010] does not influence admissibility. In fact, such pruning makes the heuristic more accurate, which means that it profits from any future research that is concerned with identifying and removing more unreachable parts of the TDG.

When using the modification-aware estimates described above, we obtain an improved variant of the MME heuristic, which we call *modification-aware TDG heuristic*, *TDG$_m$*.

Definition 4 (TDG$_\mathrm{m}$ Heuristic)**.**
Let $P = \langle PS, \prec, CL, VC\rangle$ be a partial plan. Then,

$$h_{TDG_m}(P) := \sum_{l:t(\bar{\tau})\in PS} \left(\min_{v_t \in comp(t(\bar{\tau}))} h_T(v_t) \right) - |CL|$$

5.1 Recomputing the TDG During Planning

All TDG-based search strategies or heuristics developed so far are pre-processing heuristics [Elkawkagy *et al.*, 2012; Bercher *et al.*, 2014b]. That is, they rely on a TDG that is computed only *once* – prior search. However, decompositions that are performed during search influence the TDG and, consequently, the accuracy of any heuristic that is based upon it. Therefore, we show after which plan modifications the TDG could possibly change and after which it can not.

In order to understand in which situations a recomputation of the TDG can result in a changed TDG, we need to explain the TDG construction process. We do this only *very briefly* and refer to the paper by Elkawkagy *et al.* [2010] for any further details. Starting from the given initial partial plan, all primitive tasks (i.e., actions) that are reachable via decomposition are identified. Then, using only these actions, a relaxed reachability analysis is performed. Afterwards, the TDG is constructed and limited to partial plans in which all actions are reachable via the relaxed reachability analysis.

Applying methods implies that certain actions may not be reachable via decomposition anymore. The non-availability of these actions might even prove further actions unreachable via the relaxed reachability analysis. Thus, recomputing a TDG after decompositions can improve heuristic estimates. In the following, we only explain the technique for the proposed heuristic, i.e., we assume $h = h_{TDG_c}$ – but the recomputation can also be done for the TDG$_\mathrm{m}$ heuristic.

Consider the example given in Fig. 1. When assuming $cost(p_3) = i$ and $h_M(P_{m_4}) = h_T(p_4)+h_T(a_3) = j > i$, we get $h_T(a_2) = i$. We now consider how the heuristic estimates with and without TDG-recomputation differ after the abstract task a_1 in P_{init} is decomposed. After decomposing a_1 we get two possible successor plans, P_1 and P_2. Since they both contain the same abstract task a_2, we get the same heuristic value $h(P_1) = h(P_2) = i$ without TDG-recomputation. When recomputing the TDG, we can exploit updated reachability information as follows. When assuming that the only action that enables the executability of p_3 is p_2 and further assume that p_2 cannot be reached via decomposing a_3, then the TDG for P_1 will not include the sub graph that is introduced via m_3. Thus, we get $h(P_1) = j$ and $h(P_2) = i$.

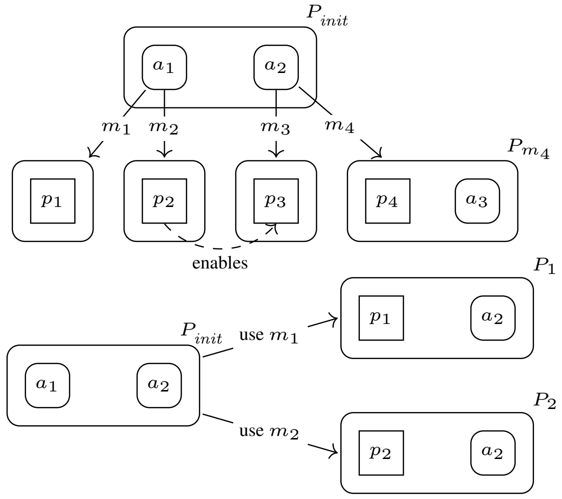


Figure 1: On top, we show a fragment of a TDG and at the bottom we show a fragment of a search space. Partial plans are denoted by surrounding boxes, abstract tasks by round boxes, and primitive tasks by square boxes.

We can thus simply recompute the TDG after each decomposition. In lifted planning, any new variable binding might also influence the set of reachable tasks, but performing the reachability analysis in all these cases turned out to be too inefficient, so we assume a ground model. So, we can set $h(P') = h(P)$ for a partial plan P and its successor P' that is obtained by a modification other than a decomposition.

Even when a decomposition has been performed, there is a special case in which we can reuse the parent node's heuristic value. This is the case if an abstract task has just a single decomposition method in its initial TDG for all its groundings, since applying such a method does not influence the set of reachable actions. In previous work, we analyzed the structural properties of the planning benchmarks that we also use in this evaluation. We observed that the average branching factor of the TDG is often smaller than 2 [Bercher *et al.*, 2014b], which means that this special case does occur in practice. Further details about how often this occurs in our problem set is given in the empirical evaluation.

Let t be the decomposed task and $v_m = \langle PS_m, \prec_m, CL_m, VC_m\rangle$ its single method vertex in the initial TDG. Due to Def. 2, we get that $h_T(t) = h_M(v_m) = \sum_{l:t'\in PS_m} h_T(t')$. Since all primitive tasks that are introduced via decomposing t were previously accounted for by the heuristic for P, but are now part of P itself and therefore contribute to its cost, we get the new heuristic value $h(P') = h(P) - \sum_{l:t'\in PS_m, t' \text{primitive}} cost(t')$.

Note that this incremental heuristic computation, which calculates a heuristic $h(P')$ based on its parent's heuristic $h(P)$, also improves runtime of the TDG-based heuristics that do not perform TDG-recomputation.

6 Evaluation

This section introduces the benchmark set, the search strategies and heuristics, and presents the empirical results.

6.1 Domains

The empirical evaluation is done based on four different HTN planning domains. Originally, these domains are hybrid planning domains. We converted them into standard HTN domains by removing preconditions and effects from the abstract tasks and by further removing all causal links that are present in the methods' partial plans.

The domains include all the ones used by Elkawkagy *et al.* [2010; 2012] and Bercher *et al.* [2014b] for their evaluations, which were also used by Alford *et al.* [2016a]. These are the *UM-Translog* domain (originally designed specifically for HTN planning), the *SmartPhone* domain (originally designed for hybrid planning), and the *Satellite* and *Woodworking* domains (both were originally designed for the International Planning Competitions (IPCs), which were adapted to hybrid planning). For further information about these four domains, we refer to previous work [Bercher *et al.*, 2014b]. In total, we have 59 problem instances. Note that all domains use unit costs. Thus, for these domains, the proposed heuristic estimates the length of a shortest solution.

6.2 Experimental Setting

We evaluate the proposed $\mathrm{TDG_c}$ heuristic in terms of its heuristic guidance power (in form of coverage) and investigate whether its admissibility creates plans of better quality for the evaluated problem set.

We implemented all strategies within the same system to allow a fair comparison. We use the hybrid planner PANDA [Bercher *et al.*, 2014b, Alg. 1], which is capable of solving HTN and hybrid planning problems. Its code will be made available online (www.uni-ulm.de/en/in/ki/panda).

Search Strategies We evaluate the blind strategies Uniform cost, Breadth First (BF), and Depth First (DF) search.

We also compare our heuristic search approach with other search techniques (i.e., systems) from the literature. We have simulated the *UMCP* algorithm [Erol *et al.*, 1994a, Fig. 2] within the used system. It always chooses some partial plan according to a plan selection function based on BF, DF, or a "heuristic" that greedily selects a partial plan with a least number of abstract tasks. In case the selected partial plan is primitive, UMCP turns it into a solution or dismisses it altogether in case this is impossible. We also include a technique for solving HTN problems that is based on a translation into classical planning [Alford *et al.*, 2016a]. Since HTN planning is more expressive than classical planning, any such translation requires a bound (for Alford *et al.*'s translation the maximum size of a task network under progression). If the resulting problem is solvable so is the original one (and a solution can be extracted from the classical solution), but if not, there might still be a solution requiring a higher bound. Alford *et al.* [2016a] provided a mechanic to compute a lower bound on the progression bound and showed empirically that the smallest bound allowing for a solution is often near this lower bound. We have therefore started with the translation using Alford *et al.*'s lower bound and increased the bound by one if no solution was found. The translated classical problems are solved by existing PDDL planners. We use the best-performing (non-optimal) planner of the evaluation by Alford *et al.* [2016a], JASPER [Xie *et al.*, 2014]. In addition, we use the (optimal) SymBAStar*-2 planner [Torralba *et al.*, 2014], the winner of the optimal track of the IPC 2014. Since the compilation creates additional actions, action costs for the original primitive tasks are unaltered and the costs for the new ones are set to zero. Note that JASPER can handle an ADL model, whereas SymBAStar*-2 handles only STRIPS. The two configurations are referred to as Compile and $\mathrm{Compile_{opt}}$, respectively. We have set the time limit for runs of the planner of each translated problem to two minutes (increasing this bound does not improve the results of the planner). The times reported are the sum of run times for all runs until a solution has been found. Please note that we did not include a comparison to SHOP2, because our models do not encode search guidance and, consequently, their decomposition methods do not specify preconditions on which the success of SHOP2 is based. For these models, SHOP2 would basically perform a blind DF progression search.

Additionally, we use both A* as well as Greedy-A*, where the heuristic is accounted for by factor 2, denoted by A_2^*. Because PANDA is a hybrid planner (i.e., it performs plan-space-based search relying on POCL concepts), we can also use the well-known POCL heuristics h_{add} and h^r_{add} [Younes and Simmons, 2003] as well as h_{relax} and h_{OC} [Nguyen and Kambhampati, 2001]. Since these heuristics are designed for (non-hierarchical) POCL problems, they only take the primitive tasks in a given partial plan into account, but make up for being less informed about the hierarchy by being extremely fast. In addition to the $\mathrm{TDG_c}$ heuristic, we evaluate our improved version of the hybrid planning heuristic MME [Bercher *et al.*, 2014b], $\mathrm{TDG_m}$. We refer to the variants with recomputation as $\mathrm{TDG_c}$-rec and $\mathrm{TDG_m}$-rec, respectively.

For our experiments, we ground the models prior search and deploy the TDG construction and domain model reduction technique by Elkawkagy *et al.* [2010].

Hardware We used a machine with Xeon E5-2660 v3 CPUs with 2.60 GHz base frequency, a memory limit of 10 GB, and a time limit of 10 minutes (CPU time) per run.

6.3 Results

Coverage Results are given in Tab. 1. We can observe that already the uninformed, blind search strategies perform impressively well with solving between 47 and 53 problems out of 59. This can in part be attributed to our grounding procedure [Elkawkagy *et al.*, 2010], which reduces the model based on the given problem instances. Bercher *et al.* [2014b] showed that almost all resulting (grounded) problem instances become acyclic.

The UMCP strategies perform similarly to the uninformed ones. They solve between 47 and 52 problems. The two configurations based on the compilation technique are performing quite differently. The compilation in combination with

Table 1: Per domain and strategy, we present the number of solved problem instances (*#s*), the number of optimally solved instances (*#o*), and the maximal plan cost over all solved problem instances relative to the optimal solution (*cost*).

	Strategy	UM-Tr. (21 inst.)			SmartPh. (5 inst.)			Satellite (22 inst.)			Woodw. (11 inst.)			Summary (59 inst.)		
		#s	#o	cost	#s	#o	cost	#s	#o	cost	#s	#o	cost	#s	#o	cost
blind	Uniform	21	21	1.00	4	4	1.00	17	17	1.00	8	8	1.00	50	50	**1.00**
	BF	21	21	1.00	4	4	1.00	15	15	1.00	7	7	1.00	47	47	**1.00**
	DF	21	21	1.00	5	1	1.60	19	7	2.09	8	4	1.44	53	33	2.09
systems	UMCP$_{BF}$	21	21	1.00	4	4	1.00	15	15	1.00	7	7	1.00	47	47	**1.00**
	UMCP$_{DF}$	21	21	1.00	4	1	1.60	17	6	2.09	6	4	1.29	48	32	2.09
	UMCP$_h$	21	21	1.00	5	4	1.40	19	11	1.50	7	7	1.00	52	43	1.50
	Compile	18	18	1.00	5	5	1.00	21	18	1.10	5	5	1.00	49	46	1.10
	Compile$_{opt}$	16	16	1.00	5	5	1.00	9	9	1.00	5	5	1.00	35	35	**1.00**
A*	ADD	21	21	1.00	4	1	1.20	21	21	1.00	10	9	1.17	56	52	1.20
	ADD-r	21	21	1.00	5	5	1.00	19	18	1.08	9	4	1.25	54	48	1.25
	Relax	21	21	1.00	5	5	1.00	18	18	1.00	10	8	1.17	54	52	1.17
	OC	21	21	1.00	4	4	1.00	21	21	1.00	10	7	1.17	56	53	1.17
	TDG$_{\mathrm{m}}$/-rec	21	21	1.00	5	5	1.00	22	21	1.31	9	9	1.00	**57**	56	1.31
	TDG$_{\mathrm{c}}$/-rec	21	21	1.00	5	5	1.00	18	18	1.00	8	8	1.00	52	52	**1.00**
A_2^*	ADD	21	21	1.00	4	0	1.20	21	20	1.09	10	9	1.17	56	50	1.20
	ADD-r	21	21	1.00	5	5	1.00	20	17	1.10	10	4	1.25	56	47	1.25
	Relax	21	21	1.00	5	5	1.00	18	15	1.10	10	4	1.25	54	45	1.25
	OC	21	21	1.00	4	4	1.00	22	21	1.09	10	7	1.22	**57**	53	1.22
	TDG$_{\mathrm{m}}$/-rec	21	21	1.00	5	5	1.00	22	17	1.31	9	8	1.08	**57**	51	1.31
	TDG$_{\mathrm{c}}$	21	21	1.00	5	5	1.00	20	20	1.00	10	10	1.00	56	56	**1.00**
	TDG$_{\mathrm{c}}$-rec	21	21	1.00	5	5	1.00	20	20	1.00	11	11	1.00	**57**	**57**	**1.00**

JASPER performs similar to the previously discussed configurations: it solves 49 problems. The compilation that uses the optimal planner SymBAStar*-2 is performing worst with solving only 35 instances. This can be attributed to two facts: First, because it is an optimal planner, which are known to incur extra search effort to guarantee optimality. Second, the compilation used for SymBAStar*-2 relies on a basic STRIPS model, which is much larger than the ADL model that is used for JASPER [Alford *et al.*, 2016a].

Concerning the evaluated heuristics, we can make several observations. First, they all perform very good, solving between 54 and 57 problems. We did not expect that the four POCL heuristics perform so impressively well, since they are completely unaware of the hierarchy and ignore all abstract tasks (this would be different if we were using the original hybrid formulations, in which also abstract tasks use preconditions and effects). Similar to the good performance of the uninformed strategies, we assume that this can in part be attributed to our grounding procedure that eliminates much of the problems' difficulty. With A*, all heuristics except for TDG$_{\mathrm{c}}$ perform similarly well. TDG$_{\mathrm{c}}$ shows the lowest coverage solving only 52 problems, which we attribute to its admissibility. Best performing is the TDG$_{\mathrm{m}}$ heuristic, which solves 57 problems. With A_2^*, TDG$_{\mathrm{m}}$ remains being among the best configurations, but additionally the TDG$_{\mathrm{c}}$ heuristic is among the best ones, which solves 56 problems when TDG-recomputation is disabled and 57 if it is enabled.

Plan Quality In Tab. 1, we give for each strategy and domain the number of problems that were solved optimal as well as the maximal solution cost relative to the optimal one. We assume that many of our problem instances only allow for solutions with the same number of actions, which makes the comparison of plan quality for these instances meaningless. We assume this being the case because for many instances, all strategies found solutions of the same quality. This is the case for all problems in UM-Translog, for 1 of 5 problems in SmartPhone, for 6 of 22 problems in Satellite, and for 2 of 11 problems in the Woodworking domain.

Table 2: Per domain, we present the number of recomputations divided by number of decompositions (*rec/dec*) and the number of improved heuristic estimates divided by number of recomputations (*h-im/rec*). For each of these values we report the minimum (*min*), maximum (*max*), and mean of means (μ).

	Strategy	rec/dec			h-im/rec			rec/dec			h-im/rec		
		min	max	μ	min	max	μ	min	max	μ	min	max	μ
		UM-Translog						SmartPhone					
A*	TDG$_{\mathrm{m}}$	.027	.188	.086	.000	.333	.032	.300	.691	.476	.000	.117	.023
	TDG$_{\mathrm{c}}$	.027	.188	.086	.000	.333	.032	.300	.647	.473	.000	.041	.008
A_2^*	TDG$_{\mathrm{m}}$	.027	.188	.086	.000	.333	.032	.300	.713	.484	.000	.121	.024
	TDG$_{\mathrm{c}}$	.027	.188	.086	.000	.333	.032	.300	.647	.471	.000	.041	.008
		Satellite						Woodworking					
A*	TDG$_{\mathrm{m}}$	.857	1.00	.956	.110	.608	.248	.294	.932	.581	.000	.548	.246
	TDG$_{\mathrm{c}}$	.750	1.00	.913	.087	.592	.264	.294	.943	.600	.000	.592	.330
A_2^*	TDG$_{\mathrm{m}}$	.857	1.00	.953	.110	.617	.268	.294	.961	.611	.000	.721	.306
	TDG$_{\mathrm{c}}$	.814	1.00	.934	.049	.609	.256	.294	.943	.615	.000	.587	.333

The overall worst-quality plans are produced by DF, which produces supoptimal solutions in 20 out of 53 cases. Their size is up to a factor of 2.09 as large as the optimal one. We can see that A* with the admissible TDG$_{\mathrm{c}}$ heuristic is the least successful A* variant in terms of coverage, but guarantees to find optimal solutions. Other heuristics sometimes find solutions of suboptimal quality, ranging up to 31%. However, in total, non-optimal solutions are only found in a few cases (see Tab. 1, summary column). E.g., TDG$_{\mathrm{m}}$, although being inadmissible, solves 56 of 57 problems optimal with A*. Using A_2^*, TDG$_{\mathrm{c}}$/-rec belong to the best-performing heuristics – solving 56 and 57 problems, respectively, all of them optimal.

TDG-Recomputation As indicated in Tab. 1, TDG$_{\mathrm{m}}$ and TDG$_{\mathrm{m}}$-rec produce exactly the same results in terms of coverage and plan quality for A* and A_2^*. The same holds for TDG$_{\mathrm{c}}$ and TDG$_{\mathrm{c}}$-rec for A*, but for A_2^*, TDG$_{\mathrm{c}}$-rec solves one problem instance more than TDG$_{\mathrm{c}}$. The impact of the recomputation can be summarized as follows: it increases heuristic accuracy resulting into more-informed heuristics (Tab. 2), which in turn results into smaller search spaces that sometimes come at the cost of increased computation times (Tab. 3) due to the overhead of recomputation.

Tab. 2 summarizes how often the TDG is recomputed and how often these recomputations are beneficial in terms of more accurate heuristic estimates. We report rec/dec, which gives the ratio of TDG-recomputations to performed decompositions. A ratio of 1 indicates that the TDG is rebuilt each time an abstract task is decomposed. We can see that in the UM-Translog domain, the recomputation is almost never performed, which goes back to the simplicity of the problem instances, where the TDG often has an average branching fac-

Table 3: For each domain, we summarize in how many problem instances the search space or time was deduced (<), unchanged (=), or increased (>), due to TDG recomputation.

Strategy		space			time			space			time			space			time			space			time		
		<	=	>	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>	<	=	>
		UM-Translog						SmartPhone						Satellite						Woodworking					
A^*	TDG_m	2	19	0	1	15	5	1	4	0	1	4	0	22	0	0	0	17	5	7	2	0	1	6	2
	TDG_c	2	19	0	0	13	8	1	4	0	0	4	1	18	0	0	0	10	8	6	2	0	4	3	1
A^*_2	TDG_m	2	19	0	3	16	2	1	4	0	0	4	1	22	0	0	0	13	9	4	5	0	1	6	2
	TDG_c	2	19	0	4	11	6	1	4	0	1	4	0	20	0	0	0	11	9	5	5	0	4	3	3

tor near 1 [Bercher *et al.*, 2014b]. All problems are solved in less than 7 s by all strategies with only little deviation between them – so this domain does not provide much insights. In the SmartPhone and Woodworking domains, recomputation is done in approximately 50% and 60% of all decompositions. In Satellite, this percentage ranges up to 95.6%. We are also interested in the fact whether these recomputations are beneficial, i.e., whether the heuristic estimates based on a recomputed TDG are more accurate than compared to using the TDG of its parent. That value is given by h-im/rec, the ratio of improved heuristic estimates to performed recomputations. A ratio of 1 means that every recomputation produced an improved heuristic value. For UM-Translog, the mean is relatively small, but this is due to the fact that only in 2 of 21 instances the recomputation improves the heuristic (in both instances, the ratio is 33.3% for all strategies). Similarly for SmartPhone: Here, the reduction increases heuristic accuracy in only one problem instance (see table for its ratio). For the others, we see a mean heuristic improvement in about 25% to 33% of all cases. This tells us that further research should investigate the issue of predicting when a recomputation will result into smaller TDGs thereby increasing the ratio h-im/rec (i.e., by reducing the number of non-beneficial recomputations). Interestingly, in nearly all cases the heuristic was increased to ∞ if it was increased at all. This means that in our problem set, if any action that belongs to the set of least-cost reachable actions becomes unreachable in the TDG, then so does every action and the entire set becomes empty.

Tab. 3 summarizes how often the recomputation pays off in terms of search space and search time. We can see that the search space never becomes larger and that the impact of recomputation depends severely on the specific domain. In the UM-Translog and Smartphone domains the search space is reduced in only a few instances, which is non-surprising given the data reported above. In the respective SmartPhone instance, the search space is reduced by 35% (both A^* and A^*_2) for TDG_m-rec and by 26% (A^*) and 28% (A^*_2) for TDG_c-rec, whereas the runtimes are unaffected compared to the runs without recomputation. In the Satellite and Woodworking domains, search space reductions occur more frequently. Even though every search space gets reduced in Satellite, these reductions are usually quite small and do therefore not pay off in terms of search time. E.g., in this domain, the highest relative runtime increase is from 10 s to 20 s with a search space reduction from 49.754 to 47.048 search nodes (5.4%). In Woodworking, the reductions are usually larger. However, also in this domain, they are not always severe enough to pay off in terms of runtime. The worst result in terms of runtime, e.g., is given in a problem instance where TDG_m reduces the search space from 121.661 nodes to 96.824 (by 20%). Due to the overhead of recomputing the TDG, the runtime increases here from 32 s to 71 s (factor 2.22). In total, the reduction pays off in the majority of all instances in terms of runtime in this domain (and it allows to solve one additional problem, not reflected in Tab. 3). In one instance, search space was reduced from 86.935 to 15.213 (by 83%) coinciding with runtime reduction from 15 s to 8 s (factor 0.46) for TDG_m and for TDG_c from 2.360 to 158 (by 93%) coinciding with runtime reduction from 5 s to 3 s (factor 0.4).

7 Summary and Conclusion

We presented one of the first admissible heuristics for hierarchical planning. Our empirical evaluation reveals that, despite its admissibility, the heuristic shows good performance in terms of coverage when deployed with Greedy-A^*. With A^*, it performs slightly worse than other evaluated heuristics in terms of coverage, but it guarantees to find optimal solutions – whereas other strategies also produce suboptimal solutions. We further proposed a technique that improves the heuristic quality. The evaluation shows that it often reduces the explored search space, but often at the cost of increased solving time. The search space reductions vary significantly between the different problem instances. The highest achieved search space reduction is 93%, which coincides with a search time reduction of 40%.

Acknowledgements

We thank the reviewers for their comments and suggestions. We also want to thank Mario Schmautz for his support with the evaluation. This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

[Alford *et al.*, 2014] Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana Nau. On the feasibility of planning graph style heuristics for HTN planning. In *ICAPS*, pages 2–10. AAAI Press, 2014.

[Alford *et al.*, 2015a] Ron Alford, Pascal Bercher, and David Aha. Tight bounds for HTN planning. In *ICAPS*, pages 7–15. AAAI Press, 2015.

[Alford *et al.*, 2015b] Ron Alford, Pascal Bercher, and David Aha. Tight bounds for HTN planning with task insertion. In *IJCAI*, pages 1502–1508. AAAI Press, 2015.

[Alford *et al.*, 2016a] Ron Alford, Gregor Behnke, Daniel Höller, Pascal Bercher, Susanne Biundo, and David Aha. Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems. In *ICAPS*, pages 20–28. AAAI Press, 2016.

[Alford *et al.*, 2016b] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. Hierarchical planning: relating task and goal decomposition with task sharing. In *IJCAI*, pages 3022–3029. AAAI Press, 2016.

[Behnke *et al.*, 2015] Gregor Behnke, Daniel Höller, and Susanne Biundo. On the complexity of HTN plan verification and its implications for plan recognition. In *ICAPS*, pages 25–33. AAAI Press, 2015.

[Bercher *et al.*, 2014a] Pascal Bercher, Susanne Biundo, Thomas Geier, Thilo Hörnle, Florian Nothdurft, Felix Richter, and Bernd Schattenberg. Plan, repair, execute, explain - how planning helps to assemble your home theater. In *ICAPS*, pages 386–394. AAAI Press, 2014.

[Bercher *et al.*, 2014b] Pascal Bercher, Shawn Keen, and Susanne Biundo. Hybrid planning heuristics based on task decomposition graphs. In *SoCS*, pages 35–43. AAAI Press, 2014.

[Bercher *et al.*, 2016] Pascal Bercher, Daniel Höller, Gregor Behnke, and Susanne Biundo. More than a name? On implications of preconditions and effects of compound HTN planning tasks. In *ECAI*, pages 225–233. IOS Press, 2016.

[Biundo and Schattenberg, 2001] Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *ECP*, pages 157–168. AAAI Press, 2001.

[Byrne, 1977] Richard Byrne. Planning meals: Problem solving on a real data-base. *Cognition*, 5:287–332, 1977.

[Elkawkagy *et al.*, 2010] Mohamed Elkawkagy, Bernd Schattenberg, and Susanne Biundo. Landmarks in hierarchical planning. In *ECAI*, pages 229–234. IOS Press, 2010.

[Elkawkagy *et al.*, 2012] Mohamed Elkawkagy, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo. Improving hierarchical planning performance by the use of landmarks. In *AAAI*, pages 1763–1769. AAAI Press, 2012.

[Erol *et al.*, 1994a] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254. AAAI Press, 1994.

[Erol *et al.*, 1994b] Kutluhan Erol, James A. Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, pages 1123–1128. AAAI Press, 1994.

[Fox, 1997] Maria Fox. Natural hierarchical planning using operator decomposition. In *ECP*, pages 195–207. Springer, 1997.

[Geier and Bercher, 2011] Thomas Geier and Pascal Bercher. On the decidability of HTN planning with task insertion. In *IJCAI*, pages 1955–1961. AAAI Press, 2011.

[Gerevini *et al.*, 2003] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs. *JAIR*, 20:239–290, 2003.

[Gerevini *et al.*, 2008] Alfonso Gerevini, Ugur Kuter, Dana S. Nau, Alessandro Saetti, and Nathaniel Waisbrot. Combining domain-independent planning and HTN planning: The Duet planner. In *ECAI*, pages 573–577. IOS Press, 2008.

[Höller *et al.*, 2014] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Language classification of hierarchical planning problems. In *ECAI*, pages 447–452. IOS Press, 2014.

[Höller *et al.*, 2016] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. Assessing the expressivity of planning formalisms through the comparison to formal languages. In *ICAPS*, pages 158–165. AAAI Press, 2016.

[Lotem *et al.*, 1999] Amnon Lotem, Dana S. Nau, and James A. Hendler. Using planning graphs for solving HTN planning problems. In *AAAI*, pages 534–540. AAAI Press, 1999.

[Marthi *et al.*, 2008] Bhaskara Marthi, Stuart Russell, and Jason Wolfe. Angelic hierarchical planning: Optimal and online algorithms. In *ICAPS*, pages 222–231. AAAI Press, 2008.

[McAllester and Rosenblitt, 1991] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *AAAI*, pages 634–639. AAAI Press, 1991.

[Nau *et al.*, 2003] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *JAIR*, 20:379–404, 2003.

[Nguyen and Kambhampati, 2001] XuanLong Nguyen and Subbarao Kambhampati. Reviving partial order planning. In *IJCAI*, pages 459–466. Morgan Kaufmann, 2001.

[Seegebarth *et al.*, 2012] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne Biundo. Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In *ICAPS*, pages 225–233. AAAI Press, 2012.

[Shivashankar *et al.*, 2016] Vikas Shivashankar, Ron Alford, Mark Roberts, and David W. Aha. Cost-optimal algorithms for planning with procedural control knowledge. In *ECAI*, pages 1702–1703. IOS Press, 2016.

[Shivashankar *et al.*, 2017] Vikas Shivashankar, Ron Alford, and David Aha. Incorporating domain-independent planning heuristics in hierarchical planning. In *AAAI*, pages 3658–3664. AAAI Press, 2017.

[Sohrabi *et al.*, 2009] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. HTN planning with preferences. In *IJCAI*, pages 1790–1797. AAAI Press, 2009.

[Torralba *et al.*, 2014] Álvaro Torralba, Vidal Alcázar, Daniel Borrajo, Peter Kissmann, and Stefan Edelkamp. SymBA*: A symbolic bidirectional A* planner. In *The 8th IPC*, pages 105–108, 2014.

[Xie *et al.*, 2014] Fan Xie, Martin Müller, and Robert Holte. Jasper: The art of exploration in greedy best first search. In *The 8th IPC*, pages 39–42, 2014.

[Younes and Simmons, 2003] Håkan L. S. Younes and Reid G. Simmons. VHPOP: Versatile heuristic partial order planner. *JAIR*, 20:405–430, 2003.

The following pages show the publication:

P. Bercher, S. Keen, and S. Biundo. “Hybrid Planning Heuristics Based on Task Decomposition Graphs”. In: *Proceedings of the 7th Annual Symposium on Combinatorial Search (SoCS 2014)*. AAAI Press, 2014, pp. 35–43

Reprinted with kind permission of AAAI Press.

The included PDF is a revised version. Modifications are not explicitly mentioned, since there were only minor corrections or improvements.

Hybrid Planning Heuristics Based on Task Decomposition Graphs

Pascal Bercher and **Shawn Keen** and **Susanne Biundo**
Institute of Artificial Intelligence,
Ulm University, Germany,
firstName.lastName@uni-ulm.de

Revised Version
(18. August 2014)

Abstract

Hybrid Planning combines Hierarchical Task Network (HTN) planning with concepts known from Partial-Order Causal-Link (POCL) planning. We introduce novel heuristics for Hybrid Planning that estimate the number of necessary modifications to turn a partial plan into a solution. These estimates are based on the task decomposition graph that contains all decompositions of the abstract tasks in the planning domain. Our empirical evaluation shows that the proposed heuristics can significantly improve planning performance.

Introduction

Hierarchical Task Network (HTN) planning relies on the concept of task decomposition (Erol, Hendler, and Nau 1994). While the goal in classical (non-hierarchical) planning is to find an action sequence that satisfies a goal description, the goal in hierarchical planning is to find an executable course of action that is a refinement of an initial partial plan. Partial plans may contain primitive and abstract tasks. While primitive tasks correspond to operators known from classical planning, abstract tasks represent complex activities and must therefore be refined (decomposed) into more concrete courses of action using so-called decomposition methods. The difficulty in solving hierarchical planning problems is to choose the "correct" decomposition method for an abstract task in a given partial plan.

To improve the performance of hierarchical planning systems, one can follow *domain-specific* approaches that encode a domain-specific search advice within the domain. SHOP2 (Nau et al. 2003) is one of the best-known hierarchical planning systems following that technique. Another approach is to design *domain-independent* search strategies (Marthi, Russell, and Wolfe 2008; Shivashankar et al. 2013; Elkawkagy et al. 2012). The hierarchical planning system GoDel (Shivashankar et al. 2013), for instance, uses landmarks known from classical planning (Porteous, Sebastia, and Hoffmann 2001) for search guidance. The hierarchical planning system by Elkawkagy et al. (2012) uses *hierarchical landmarks* to guide its search. These are abstract and primitive tasks, which occur on any refinement process from the initial partial plan to any solution plan. Such hierarchical landmarks can be extracted from a task decomposition graph (TDG), which represents the decomposition hierarchy of the planning problem at hand. They use these landmarks in order to decide whether one decomposition method is preferred over another. For the selection of a most-promising search node, however, conventional heuristics are used that are unaware of the underlying hierarchy.

In this paper, we introduce novel heuristics that *do* take hierarchical information into account. We use the Hybrid Planning paradigm (Kambhampati, Mali, and Srivastava 1998; Biundo and Schattenberg 2001) that fuses hierarchical planning with concepts known from Partial-Order Causal-Link (POCL) planning. We propose a novel variant of the *Hierarchical Decomposition Partial-Order Planner* HD-POP (Russell and Norvig 1994, edition 1, p. 374–375) suited for Hybrid Planning. The resulting system, PANDA, guides its search using informed heuristics. We propose such heuristics that estimate the number of modifications necessary to find a solution. To that end, the TDG is used to extract hierarchical landmarks and further information about the hierarchy. Our empirical evaluation shows that the proposed heuristics can significantly improve planning performance.

Hybrid Planning

Planning problems are given in terms of a *Hybrid Planning* formalization (Kambhampati, Mali, and Srivastava 1998; Biundo and Schattenberg 2001), which fuses concepts from Hierarchical Task Network (HTN) planning (Erol, Hendler, and Nau 1994) with concepts from Partial-Order Causal-Link (POCL) planning (McAllester and Rosenblitt 1991; Penberthy and Weld 1992).

In Hybrid Planning, there are two kinds of tasks: *primitive* and *abstract* tasks. Both primitive and abstract tasks $t(\bar{\tau})$ are tuples $\langle prec(\bar{\tau}), \mathit{eff}(\bar{\tau})\rangle$ consisting of a *precondition* and *effect* over the task parameters $\bar{\tau}$. Preconditions and effects are conjunctions of literals. As usual, states are sets of ground atoms. Tasks are called ground if all variables are bound to constants. Applicability of (sequences of) ground primitive tasks is defined as usual.

Partial plans are tuples $(PS, \prec, VC, CL)$ consisting of the following elements. The set of plan steps PS is a set of uniquely labeled tasks $l : t(\bar{\tau})$. The set $\prec$ of ordering constraints induces a partial order on the plan steps in PS. The set VC is a set of variable constraints that (non-)codesignate the task parameters with each other or with constants. CL is a set of causal links. A causal link $l:t(\bar{\tau}) \rightarrow_{\varphi(\tau_i)} l':t'(\bar{\tau}')$ denotes that the precondition literal $\varphi(\tau_i)$ of the plan step $l':t'(\bar{\tau}')$ is supported by the same effect of the plan step $l:t(\bar{\tau})$. If there is no causal link to a precondition $\varphi(\tau_i)$ we call it an *open* precondition.

Partial plans may also contain abstract tasks. These cannot be executed directly. Instead, they need to be decomposed into more specific partial plans using so-called (decomposition) methods. A method $m = \langle t(\bar{\tau}), P \rangle$ maps an abstract task $t(\bar{\tau}) = \langle prec(\bar{\tau}), \mathit{eff}(\bar{\tau}) \rangle$ to a partial plan P that "implements" that task (Biundo and Schattenberg 2001). Thereby, causal links involving $t(\bar{\tau})$ can be passed down to one of its sub tasks within P when decomposing $t(\bar{\tau})$.

Now, a *Hybrid Planning Domain* $\mathcal{D}$ is given by the tuple $\langle T_a, T_p, M \rangle$ consisting of a finite set of abstract and primitive tasks T_a and T_p, respectively, and a set of methods M. A *Hybrid Planning Problem* is given by a domain and an initial partial plan P_{init}. As it is the case in standard POCL planning, P_{init} contains two special primitive tasks that encode an initial state and a goal description. The task t_0 has no precondition and the initial state as effect. The task t_∞ has the goal description as precondition and no effects.

A plan P_{sol} is a solution to a hybrid planning problem if and only if the following criteria are met:

1. P_{sol} is a refinement of P_{init} w.r.t. the decomposition of abstract tasks and insertion of ordering constraints, variable constraints, and causal links.
2. P_{sol} needs to be executable in the initial state. Thus,
   (a) all tasks are primitive and ground,
   (b) there are no open preconditions, and
   (c) there are no causal threats. That is, given a causal link $l:t(\bar{\tau}) \rightarrow_{\varphi(\tau_i)} l':t'(\bar{\tau}')$, we call the task $l'':t''(\bar{\tau}'')$ a threat to that link if and only if the ordering constraints allow it to be ordered between the tasks of the causal link and it has an effect $\neg\psi(\tau_i'')$ that can be unified with the protected condition $\varphi(\tau_i)$.

Criterion 1 relates any solution plan to the initial partial plan P_{init}. This is necessary, since P_{init} represents the actual planning problem. Note that one could also allow the insertion of tasks into a partial plan without being introduced via decomposition of an abstract task as one of the allowed refinement options. In this paper, however, we do not allow such insertions and thereby follow the typical HTN planning approach (Erol, Hendler, and Nau 1994). Other hierarchical planning approaches, such as the Hybrid Planning approach by Kambhampati et al. (1998) or HTN Planning with Task Insertion (Geier and Bercher 2011) do allow such insertions.

Criterion 2 is inherited from standard POCL planning. It ensures that any linearization of the tasks in a solution plan is executable in the initial state. Every linearization generates a state which satisfies the goal condition.

Planning Algorithm

We search for solutions by systematically refining the initial partial plan P_{init} until it meets all the solution criteria. To that end, we propose the following generic hybrid planning algorithm (cf. Alg. 1). The corresponding planning system PANDA (Planning and Acting in a Network Decomposition Architecture) is based on earlier work (Schattenberg 2009).

Algorithm 1: Hierarchical Refinement Planning

```
1 $F \leftarrow \{P_{init}\}$
2 while $F \neq \emptyset$ do
3     $P \leftarrow$ planSel($F$)
4     $F \leftarrow F \setminus \{P\}$
5     if Flaws($P$) $= \emptyset$ then return $P$
6     $f \leftarrow$ flawSel(Flaws($P$))
7     $F \leftarrow F \cup \{$ modify($m, P$) $\mid m \in$ Mods($f, P$) $\}$
8 return fail
```

The algorithm employs search in the space of partial plans. It uses a search fringe F consisting of all created refinements of P_{init} that have not yet been chosen for expansion. First, it picks and removes a partial plan from the fringe F (line 3, 4). This choice is done by means of the *plan selection function* planSel. This function can be implemented in various ways and determines the actual search strategy. For instance, choosing always an "oldest" plan results in a *breadth first search*. More elaborated strategies like A^* or *greedy search* need heuristic functions to judge the quality or goal distance of partial plans, such as the ones we propose in this paper.

After a partial plan P has been chosen for refinement, we calculate all its *flaws* Flaws(P). Flaws are syntactical representations of violations of solution criteria. For instance, every abstract task in P induces a so-called *abstract task flaw*, since its existence violates the executability required by solution criterion 2.(a). Further flaw classes are *open precondition flaws* and *causal threat flaws* according to solution criteria 2.(b) and 2.(c), respectively.

If P has no flaws, it is a solution and hence returned (line 5). If there are flaws, they need to be removed in a systematic manner. We follow the approach of the hierarchical planner *HD-POP* (Russell and Norvig 1994, edition 1, p. 374–375) and pick *one* of its flaws (line 6) to be resolved. To that end, the function Mods(f, P) calculates all *modifications* that modify the partial plan P, such that the flaw f is addressed in the resulting refinement. Modifications specify which plan elements are to be added to or removed from a given partial plan (Schattenberg, Bidot, and Biundo 2007). The application of a modification m to the partial plan P by the function modify(m, P) generates the corresponding successor plan. Abstract task flaws can only be resolved by applying a decomposition method for the respective task. Open precondition flaws can only be resolved by inserting a causal link or by decomposing an abstract task that might possibly introduce a task with a suitable effect. Finally, a causal threat flaw can be resolved by promotion, demotion, and separation, as well as by decomposition if one of the involved tasks is abstract.

The selection of a specific flaw does not constitute a backtracking point. This is due to the fact that every flaw needs to be resolved at some point and we generate *all* its possible successor plans (line 7). Hence, any partial plan can be discarded if there is a flaw without modifications. Although the choice of a flaw does not influence correctness or completeness, it heavily influences planning performance. This choice point is one of the major differences to the hierarchical planner used by Elkawkagy et al. (2012). That planner orders all modifications and therefore comes without an explicit choice point for flaws.

After all successor plans of the selected partial plan and flaw have been inserted into the fringe, the loop starts over until a solution is generated or the fringe becomes empty. In case of an empty fringe, there is (provably) no solution and *fail* is returned (line 8).

Exploiting Task Decomposition Graphs

We now give some basic definitions based on TDGs that can be used to design heuristic functions.

Let $\mathcal{G} = \langle V_T, V_M, E\rangle$ be a TDG in accordance to Def. 5 given by Elkawkagy et al. (2012). Hence, $\mathcal{G}$ is a directed AND/OR graph with the following elements: V_T is a set of task vertices consisting of ground abstract and primitive tasks that can be obtained by decomposing the initial partial plan. V_M is a set of method vertices consisting of ground methods that decompose an abstract task within V_T. E is a set of edges connecting vertices from V_T with vertices from V_M and vice versa. More precisely: If $t(\bar{c}) \in V_T$ and $m = \langle t(\bar{c}), P\rangle \in V_M$, then $(t(\bar{c}), m) \in E$. Then, the method node m has an edge for every one of the ground plan steps in P to its respective tasks in V_T: If $t'(\bar{c}') \in V_T$, $t'(\bar{c}')$ being a task of the plan steps in P, then $(m, t'(\bar{c}')) \in E$. Fig. 1 shows a small example TDG.

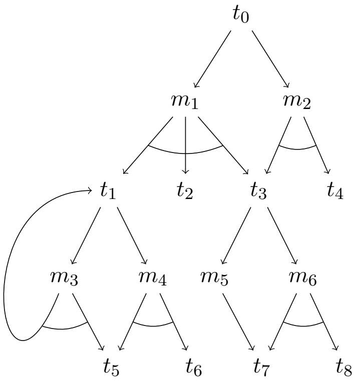


Figure 1: Example TDG depicted as AND/OR graph. The symbols t_0, t_1, t_3 and $t_2, t_4, \ldots, t_8$ represent ground abstract and ground primitive task vertices, respectively. The symbols m_1 through m_6 depict method vertices for the abstract tasks t_0, t_1, and t_3.

Although TDGs are finite, they can still be quite large because of the huge number of ground task and method instances that are possible in general. To be able to build TDGs despite their potentially large size, we follow a technique that allows to ignore certain irrelevant parts of the TDG (Elkawkagy, Schattenberg, and Biundo 2010).

Some of our TDG-based estimates are based on the concept of mandatory tasks. They have been introduced by Elkawkagy et al. (2012, Def. 6) and serve as approximation for landmarks (cf. Def. 4). Mandatory tasks are ground tasks that occur in all decomposition methods of the same abstract task. Elkawkagy et al. need these tasks as an intermediate step to calculate the set of optional tasks. In their algorithm, they use the number of these optional tasks to decide whether one decomposition method is preferred before another. We only require the mandatory tasks and use them to obtain an estimate for the modifications that need to be performed when decomposing abstract tasks. In contrast to Elkawkagy et al. (2012), we do not only count the tasks, but also use the number of their preconditions for our estimates.

To define mandatory tasks, we first need to define the *sub task set* $S(t(\bar{c}))$ of a ground abstract task $t(\bar{c})$, $\bar{c}$ denoting a sequence of constants the task's parameters are codesignated with. For each method m for a task $t(\bar{c})$, $S(t(\bar{c}))$ contains a set with all tasks in the partial plan referenced by m:

$$S(t(\bar{c})) = \{\{t'(\bar{c}') \mid (m, t'(\bar{c}')) \in E\} \mid (t(\bar{c}), m) \in E\}$$

For the TDG in Fig. 1, $S(t_0) = \{\{t_1, t_2, t_3\}, \{t_3, t_4\}\}$, $S(t_1) = \{\{t_1, t_5\}, \{t_5, t_6\}\}$, and $S(t_3) = \{\{t_7\}, \{t_7, t_8\}\}$.

The set of mandatory tasks $M(t(\bar{c}))$ of a ground abstract task $t(\bar{c})$ is then given by:

$$M(t(\bar{c})) = \bigcap_{s \in S(t(\bar{c}))} s$$

The set $M(t(\bar{c}))$ contains only tasks that inevitably will be inserted into a partial plan when decomposing $t(\bar{c})$. It therefore serves as a lower bound for estimating the modification effort for decomposing $t(\bar{c})$. In our example, we have $M(t_0) = \{t_3\}$, $M(t_1) = \{t_5\}$, and $M(t_3) = \{t_7\}$.

So far, we only consider the very next level of decomposition. Recursively incorporating the next levels for all tasks in $M(t(\bar{c}))$ results in higher estimates which are still lower bounds on the actual modification effort. The resulting set, the closure of $M(t(\bar{c}))$, denoted as $M^*(t(\bar{c}))$ is given by the following recursive equation having $M^*(t(\bar{c})) = M(t(\bar{c})) = \emptyset$ for a primitive task $t(\bar{c})$.

$$M^*(t(\bar{c})) = M(t(\bar{c})) \ \cup \bigcup_{t'(\bar{c}') \in M(t(\bar{c}))} M^*(t'(\bar{c}'))$$

Note that $M^*(t(\bar{c}))$ is finite even if the underlying TDG is cyclic. In our example, we have $M^*(t_0) = \{t_3, t_7\}$, $M^*(t_1) = M(t_1)$, and $M^*(t_3) = M(t_3)$.

The set $M^*(t(\bar{c}))$ can now be used to estimate the effort of decomposing $t(\bar{c})$. The easiest way is to take its cardinality, because for each task in that set, the planner needs to apply at least one modification (a task $t'(\bar{c}') \in M^*(t(\bar{c}))$ needs to be decomposed if $t'(\bar{c}')$ is abstract and – assuming primitive tasks have non-empty preconditions – at least one causal link must be inserted if $t'(\bar{c}')$ is primitive).

Definition 1 (Task Cardinality) *Let $t(\bar{c})$ be a ground abstract task. Then, the task cardinality of $t(\bar{c})$ is given by:*

$$TC(t(\bar{c})) := |M^*(t(\bar{c}))|$$

As argued before, $TC(t(\bar{c}))$ can be regarded as a reasonable estimate for the decomposition effort of $t(\bar{c})$. However, we can improve that estimate by taking their preconditions into account. We know that every precondition of every task needs to be supported with a causal link. Hence, we can assume that the number of required modifications is at least as large as the number of preconditions of the mandatory tasks.

Definition 2 (Precondition Cardinality) *Let $t(\bar{c})$ be a ground abstract task. Then, the precondition cardinality of $t(\bar{c})$ is given by:*

$$PC(t(\bar{c})) := \sum_{\substack{t'(\bar{c}') \in M^*(t(\bar{c})),\ t'(\bar{c}')\ \mathit{primitive}, \\ \mathit{and}\ t'(\bar{c}') = \langle \mathit{prec}(\bar{c}'), \mathit{eff}(\bar{c}') \rangle}} |\mathit{prec}(\bar{c}')|$$

Note that we only incorporate the preconditions of the primitive tasks to avoid counting preconditions twice: since causal links to or from an abstract task are handed down to their subtasks when decomposing that task, it suffices to incorporate the preconditions of their primitive sub tasks.

So far, we have introduced two estimates for the modification effort of abstract tasks by focusing on the mandatory ones. Only focusing on these tasks might be too defensive, however. Consider a planning problem, where each abstract task has at least two (ground) decomposition methods, but only a few mandatory tasks. Our estimate does not consider the remaining tasks and could therefore still be improved. In our example, $M^*(t_0)$ ignores the tasks t_1, t_2, and t_4, although at least one of these is introduced when decomposing t_0 (they could be considered disjunctive landmarks). We hence investigate a third estimate that judges the minimal modification effort based on the entire TDG while combining the ideas of TC and PC.

Definition 3 (Minimal Modification Effort) *Let V be an arbitrary set of ground tasks. For a primitive task $t(\bar{c}) = \langle \mathit{prec}(\bar{c}), \mathit{eff}(\bar{c}) \rangle$, we set $h(t(\bar{c}), V) := |\mathit{prec}(\bar{c})|$.*

For an abstract task $t(\bar{c}) = \langle \mathit{prec}(\bar{c}), \mathit{eff}(\bar{c}) \rangle$ we set:

$$h(t(\bar{c}), V) := \begin{cases} 1 + |\mathit{prec}(\bar{c})| & \text{if } t(\bar{c}) \in V, \text{ or else:} \\ 1 + \min\limits_{s \in S(t(\bar{c}))} \sum\limits_{t'(\bar{c}') \in s} h(t'(\bar{c}'), \{t(\bar{c})\} \cup V) & \end{cases}$$

Then, $\mathit{MME}(t(\bar{c})) := h(t(\bar{c}), \emptyset)$.

The basic idea of MME is to minimize the estimated modification effort per decomposition method represented by the different sets in $S(t(\bar{c}))$. The effort for a set of tasks within the same method is obtained by summing over the efforts for the single tasks. If such a task is primitive, we take the number of its preconditions as estimate, analogously to the estimate PC. If such a task is abstract, we account for its decomposition effort by 1 plus the effort of its "cheapest" decomposition method. Since the traversed TDG might be cyclic, we need to ensure termination. For that purpose, we use a set V of already visited tasks. If we discover an abstract task that was already decomposed and hence within V, we estimate its modification effort by 1 (for its decomposition) plus the number of its preconditions.

MME shows some similarities to the *add heuristic* (Bonet and Geffner 2001) that estimates the number of actions required to achieve an atom. The add heuristic, however, is a non-admissible heuristic w.r.t. the number of actions (due to the assumption of sub goal independence), while MME is admissible w.r.t. the number of modifications (while dominating both TC and PC).

Please note that not only the TDG can be calculated in a preprocessing step before the actual search, but also the functions we have presented so far (Def. 1 to 3). Thus, during search, no complicated calculations are necessary.

We still have to cope with the problem that the TDG consists entirely of ground task instances while the actual search process is done in a lifted fashion, where tasks are only partially ground. Given a partial plan $P = (PS, \prec, VC, CL)$ and one of its plan steps $ps = l : t(\bar{\tau})$, we define $\mathit{Inst}(ps)$ as a function that maps to a fixed, but arbitrary ground instance $t(\bar{c})$ of ps that is compatible with the variable constraints VC. During search, we then use the task $\mathit{Inst}(ps)$ when we want to retrieve an estimate for $t(\bar{\tau})$ from the TDG. An alternative would be to minimize or aggregate all (or some) of the possible ground instances of $t(\bar{\tau})$. We did not yet evaluate these possibilities, however.

Heuristic Functions

We want to estimate the number of necessary modifications to turn a partial plan P into a solution. In standard POCL planning, the heuristics for that purpose (Nguyen and Kambhampati 2001; Younes and Simmons 2003; Bercher, Geier, and Biundo 2013; Bercher et al. 2013) are based on the concept of task insertion, rather than task decomposition and hence not directly applicable in our setting. There are also several heuristics for the hybrid planning approach (Schattenberg, Bidot, and Biundo 2007; Schattenberg 2009). However, these heuristics incorporate the hierarchical aspects of abstract tasks not to their full extent, in particular, because they are not based upon a TDG.

A relatively simple hybrid and/or POCL heuristic is $h_{\#F}$ that returns the number of flaws. That heuristic might underestimate the number of required modifications, however. For example, estimating the number of modifications for an abstract task flaw by one ignores that decomposing the respective task introduces several new tasks thus raising new flaws. We hence estimate the number of required modifications by $h_{\#F}$ plus a value accounting for the hierarchy by inspecting the TDG w.r.t. the abstract tasks in a partial plan.

We now give heuristics for partial plans based on Def. 1 to 3, i.e., TC, PC, and MME. For all definitions, assume P to be a partial plan $(PS, \prec, VC, CL)$.

Definition 4 (Heuristic h_{TC+PC})

$$h_{TC+PC}(P) := \sum_{ps \in PS} TC(\mathit{Inst}(ps)) + PC(\mathit{Inst}(ps))$$

While both $TC(t(\bar{c}))$ and $PC(t(\bar{c}))$ guarantee not to overestimate the number of modifications for $t(\bar{c})$, h_{TC+PC} does not show this property. First, the arbitrary compatible instantiation of the plan step ps can be suboptimal. Second, TC, which is simply the cardinality of *all* the mandatory tasks of $Inst(ps)$, contributes to the modification effort for each primitive task by one, assuming that for every primitive task, at least one causal link must be inserted. This effort, however, is already reflected in $PC(Inst(ps))$, as it counts the preconditions of the primitive tasks. We did not "fix" this possible overestimation, because we believe that the heuristics already underestimate the actual modification effort in practice (most importantly because both TC and PC restrict their estimates to the mandatory tasks).

The next heuristic, based on MME, incorporates the whole TDG for its estimates.

Definition 5 (Heuristic h_{MME})

$$h_{MME}(P) := \sum_{ps \in PS \text{ is abstract}} MME(Inst(ps))$$

Evaluation

We evaluate the proposed TDG-based heuristics on four hybrid planning domains and compare them with standard search strategies that are not based on the TDG.

Planning Benchmarks

We use the same set of planning domains used by Elkawkagy et al. (2012) for their evaluation. We shortly review those domains and include information about the TDGs of the evaluated problem instances. We only build the *pruned* TDGs that incorporate reachability information of the respective problem instances. One of the TDG-related information is its maximal branching factor. Please note that this number is different from the branching factor of the explored search space, since not only decomposition methods need to be chosen, but also open precondition and causal threat flaws need to be resolved. In particular for inserting a causal link there might be several refinement options during search.

The first planning domain is based on the well-known **UM-Translog** (Andrews et al. 1995) domain for hierarchical planning. UM-Translog is a logistics domain, where goods of certain kinds need to be transported. It shows 48 primitive and 21 abstract tasks for which there are 51 methods. The domain contains recursive decompositions. In the conducted experiments, the respective TDGs are acyclic and contain 7 to 22 ground primitive tasks and 12 to 30 ground abstract tasks. The longest path within the TDG has length 11, while the average branching factor of the abstract tasks is at most 1.11 indicating that the reachability analysis ruled out most of the available decomposition options. The easiest problem instance can be solved by applying 25 modifications, while the hardest instance requires at least 76 modifications.

The **Satellite** domain is a hierarchical adaptation of a domain taken from the International Planning Competition (IPC) that features conducting orbital observations using a certain number of satellites and modes. It consists of 5 primitive and 3 abstract tasks with 8 methods in total. That domain does also not contain recursive decompositions. Despite the small number of tasks and the missing recursion, the domain is rather difficult due to the large number of reachable ground task instances. Those vary from 7 to 87 for primitive tasks and from 4 to 16 for abstract tasks. While the maximal path length of these TDGs is always 4, the average branching factor for decomposition ranges from 2.75 to 15.1. Depending on the problem instance, solutions require between 13 and 41 modifications.

The **SmartPhone** domain models a modern cell phone. The tasks are concerned with sending messages and creating contacts or appointments. It is defined over 87 primitive tasks, 50 abstract tasks, and 94 methods. The domain allows for recursion. The TDGs contain between 10 and 19 primitive ground tasks and 7 and 22 ground abstract tasks. The maximal acyclic path length ranges from 4 to 6 with an average branching factor ranging from 1.42 to 1.95. Solutions have a minimal depth of 18 to 54.

The last domain, **Woodworking**, is also based on a benchmark from the IPC and deals with cutting, planing, and finishing wood parts. The domain consists of 6 abstract tasks, 13 primitive tasks, and 14 methods. The domain does not have cyclic method definitions. The TDGs of the problem instances contain between 10 and 64 ground primitive tasks and 15 to 492 ground abstract tasks. The depth varies from 2 to 4, and the average branching factor ranges from 2.53 to 7.21. Solutions require between 22 and 53 modifications.

Search Strategies

For the evaluation, we use greedy search with varying heuristics. If some partial plans show the same heuristic value, ties are broken by chance.

For the flaw selection function, we always use Least-Cost Flaw-Repair (LCFR) (Joslin and Pollack 1994). LCFR minimizes the branching factor per search node by selecting a flaw that has minimal "repair cost", i.e., the least number of modifications. Ties between flaws are broken by chance.

Besides our new TDG-based heuristics, we have also included the *Number of Flaws* ($h_{\#F}$) of a partial plan and the *Number of Modifications* ($h_{\#M}$) for all flaws[1].

For every heuristic, we also evaluated a normalized version thereof. Let $P = (PS, \prec, VC, CL)$ be a partial plan; then $\|h(P)\|$ is defined by $\frac{h(P)}{|PS|}$. Taking the ratio of a heuristic (such as the number of flaws) to the number of plan steps prevents heuristic values for two consecutive partial plans from jumping too much. Consider two partial plans, P_1 and P_2, P_2 being the successor of P_1 due to the decomposition of an abstract task. In general, P_2 contains several new plan steps and therefore several new flaws. According to the heuristic $h_{\#F}$, for example, P_2 looks much worse than P_1 although the decomposition generating P_2 was inevitable. Normalizing tries to compensate that phenomenon.

In addition to greedy search using different heuristics, we also include several base line configurations. These include the uninformed *Breadth First Search* (BF) and *Depth First Search* (DF). Furthermore, the core ideas behind the well-known hierarchical planning systems UMCP (Erol, Hendler, and Nau 1994) and SHOP2 (Nau et al. 2003) can be captured by our hybrid planning framework (Schattenberg, Weigl, and Biundo 2005).

[1] Greedy search using $h_{\#M}$ corresponds to the search strategy used in the evaluation by Elkawkagy et al. (2012). They called this strategy *Fewer Modifications First* (fmf).

In case of UMCP, our emulation is very close to the original system. We have employed all three variants of that system described in Erol's dissertation (1996, Chapter 6.1.2.1). As proposed by Erol, we employ BF and DF as plan selection, as well as greedy search using a heuristic that always selects a partial plan with the smallest number of abstract tasks. UMCP always decomposes an abstract task before resolving open precondition flaws or causal threats.

To emulate SHOP2, we employ a depth first search with the flaw selection *earliest first*. That function always prefers a flaw associated with a plan element that is closest to the initial task, i.e., closest to the execution horizon. Note that our emulation of SHOP2 still shows some significant differences to the actual SHOP2 system. SHOP2 performs progression search in the space of states while our system is based on POCL techniques. Furthermore, the actual SHOP2 system uses domain-specific search space information that is encoded in the preconditions of methods while our approach uses domain-independent search strategies.

System Configuration

In all experiments – including the ones for SHOP2 and UMCP – we enable the TDG pruning technique described by Elkawkagy et al. (2010). The explored search space is reduced by omitting decomposition methods that are not supported by the pruned TDG.

We conducted the experiments on a system with Intel Xeon E5 2.5 Ghz processors. Because most of the problem instances are solved within only a few seconds, we report and focus on the number of expanded search nodes to obtain a more accurate measure of search effort. Note that the number of *created* search nodes is in general much larger. We limited the available CPU time per problem instance to 10 minutes and the available memory to 2 GB.

To obtain statistically significant values, we run each experiment 50 times and report the (rounded) mean number of expanded search nodes μ_s, its relative standard deviation σ/μ_s, and the (rounded) mean CPU time μ_t in seconds, including preprocessing. We report the number of successful runs in parentheses if some were not successful. The reported values are based only on these successful runs.

Experimental Results

The results for the evaluated domains are shown in Tab. 1 to 4. The best result is highlighted in bold and the second best in bold and italic.

UM-Translog In this domain, we did not find any significant differences between the compared configurations.

The evaluated UM-Translog problem instances[2] turn out to be very easy, since even the uninformed BF and DF strategies always found a solution very quickly. We attribute this result to the TDG pruning that eliminates most of the choices for different decomposition methods. Remember that the pruned TDG has a branching factor of at most 1.11 in this domain. Despite that observation, the experiments show that the evaluated UMCP and SHOP2 configurations performed much worse than the other strategies. Concerning the other configurations, results have to be interpreted with care, since the results do not differ significantly (cf. Tab. 1). We observe that in all but one problem instances, greedy search using $h_{\#F}+h_{TC+PC}$ and $h_{\#F}+h_{MME}$, respectively, was among the two best-performing configurations. Another interesting result is that both $h_{\#F}+h_{TC+PC}$ and $h_{\#F}+h_{MME}$ dominate the heuristic $h_{\#F}$ in *all* problem instances. We can hence conclude that adding the estimates based on the TDG improves the estimate that is entirely based on $h_{\#F}$.

Satellite While the small problem instances can be solved almost instantly, the problem instances in which three observations have to be taken require more than 100.000 node expansions for many of the configurations. In this domain, all our TDG-based heuristics appear to be quite well informed. In 8 of 15 problem instances, $\|h_{\#F}+h_{TC+PC}\|$ and $\|h_{\#F}+h_{MME}\|$ are the two best-performing heuristics. We can also observe that these normalized versions of our heuristics expand less nodes than their non-normalized versions in all but a few cases. In 12 of the 15 evaluated problem instances, one of the proposed four heuristics is among the two best-performing configurations. Our heuristics are in general performing very well in this domain. Especially in comparison to SHOP2 and UMCP, a much better search efficiency can be observed for the proposed heuristics. In some problem instances, for example in 3–1–1, 3–2–1, and 3–3–1, the proposed heuristics expand several hundred thousand search nodes less than the SHOP2 and the BF and DF versions of UMCP. The heuristic version of UMCP, however, works very well in that domain, but is still dominated by our TDG-based heuristics in many cases.

Since the Satellite domain shows the largest deviation among the different search strategies and heuristics, we have included a plot (Fig. 2(a)) showing the number of solved instances over the number of search node expansions. Higher curves indicate that more solutions were found given the same number of expansions. As a baseline, we included BF, DF, and the SHOP2 configuration as well as UMCP-H, which is the best-performing variant of UMCP in that domain, and $h_{\#M}$. Concerning our proposed heuristics, we included $\|h_{\#F}+h_{MME}\|$ in the plot, as it is the one with the best results. The heuristic $\|h_{\#F}+h_{TC+PC}\|$ shows a similar behavior: its graph lies only slightly below that of $\|h_{\#F}+h_{MME}\|$. We also included the non-normalized version of that heuristic, $h_{\#F}+h_{MME}$. Again, the graphs of $h_{\#F}+h_{MME}$ and $h_{\#F}+h_{TC+PC}$ are almost identical.

The plot clearly reveals the superiority of the normalized versions compared to their non-normalized counterparts. Furthermore, the best configuration in that domain is the one based on $\|h_{\#F}+h_{MME}\|$. Also, UMCP-H and DF both perform very well in that domain.

Concerning CPU time, we also produced a plot corresponding to Fig. 2(a) where the x-axis shows the CPU time.

[2] We evaluated 20 problem instances while we report only 8 of these in Tab. 1. The remaining instances turned out to be uninteresting for our evaluation, since all tested search strategies produce the same number of search nodes with a standard deviation of 0. Expanded search nodes vary from 25 to 55.

Table 1 to 4: **Strategy** lists the used system configuration in the first six cases. In the remaining cases, it specifies the heuristic used with greedy search and the flaw selection LCFR. **Problem** lists the used problem instances. The column μ_s reports the rounded mean number of expanded search nodes over 50 runs, σ/μ_s its relative standard deviation, and μ_t the rounded mean CPU time in seconds. The number of successful runs is reported in parentheses if it is not 50.

Table 1: Results for the **UM-Translog** domain.

Problem	**#06**			**#08**			**#09**			**#10**			**#11**			**#12**			**#13**			**#14**		
Strategy	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	71	0.06	**0**	93	0.29	**0**	765	0.54	5	**34**	0.00	**0**	**33**	0.00	**0**	85	0.03	**1**	77	0.03	**1**	78	0.03	**1**
DF	62	0.14	**0**	**35**	0.00	**0**	**76**	0.02	***2***	**34**	0.00	**0**	**33**	0.00	**0**	***80***	0.05	**1**	***73***	0.04	**1**	***73***	0.05	**1**
SHOP2	21121	2.43	6	**35**	0.00	**0**	***80***	0.37	***2***	**34**	0.01	**0**	**33**	0.00	**0**	94	0.25	**1**	**71**	0.08	**1**	83	0.28	**1**
UMCP-BF	205	0.19	***1***	84	0.04	**0**	1548	0.08	8	36	0.02	**0**	**33**	0.02	**0**	174	0.14	3	266	0.13	6	168	0.16	3
UMCP-DF	119	0.57	***1***	**35**	0.01	**0**	542	0.74	4	***35***	0.03	**0**	**33**	0.02	**0**	133	0.33	***2***	168	0.47	4	126	0.32	***2***
UMCP-H	93	0.55	**0**	***38***	0.14	**0**	315	1.08	4	**34**	0.02	**0**	**33**	0.02	**0**	99	0.29	**1**	132	0.49	3	90	0.25	**1**
$h_{\#M}$	58	0.10	**0**	**35**	0.00	**0**	81	0.07	3	**34**	0.00	**0**	**33**	0.00	**0**	84	0.02	***2***	75	0.02	***2***	77	0.02	***2***
$\lVert h_{\#M} \rVert$	56	0.10	**0**	**35**	0.00	**0**	**76**	0.02	3	**34**	0.00	**0**	**33**	0.00	**0**	84	0.03	***2***	76	0.02	***2***	77	0.03	***2***
$h_{\#F}$	58	0.08	**0**	39	0.13	**0**	96	0.17	3	**34**	0.00	**0**	**33**	0.00	**0**	84	0.02	**1**	76	0.03	***2***	77	0.02	***2***
$\lVert h_{\#F} \rVert$	58	0.08	**0**	**35**	0.00	**0**	**76**	0.02	***2***	**34**	0.00	**0**	**33**	0.00	**0**	83	0.03	**1**	75	0.02	***2***	77	0.03	***2***
$h_{\#F} + h_{TC+PC}$	**54**	0.05	**0**	**35**	0.00	**0**	**76**	0.02	**1**	**34**	0.00	**0**	**33**	0.00	**0**	**78**	0.03	**1**	75	0.03	***2***	**71**	0.03	**1**
$\lVert h_{\#F} + h_{TC+PC} \rVert$	57	0.11	**0**	**35**	0.00	**0**	**76**	0.03	**1**	**34**	0.00	**0**	**33**	0.00	**0**	***80***	0.04	**1**	75	0.03	***2***	***73***	0.04	**1**
$h_{\#F} + h_{MME}$	***55***	0.07	**0**	**35**	0.00	**0**	**76**	0.02	**1**	**34**	0.00	**0**	**33**	0.00	**0**	**78**	0.04	**1**	75	0.03	***2***	**71**	0.04	**1**
$\lVert h_{\#F} + h_{MME} \rVert$	64	0.12	***1***	**35**	0.00	**0**	**76**	0.03	**1**	**34**	0.00	**0**	**33**	0.00	**0**	81	0.04	**1**	76	0.03	***2***	74	0.04	**1**

Table 2: Results for the **Satellite** domain. The caption *X–Y–Z* stands for X observations, Y satellites, and Z modes.

Problem	**1–1–1**			**1–2–1**			**2–1–1**			**2–1–2**			**2–2–1**			**2–2–2**			**3–1–1**			**3–1–2**		
Strategy	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	21	0.18	**0**	44	0.10	**0**	292	0.22	***1***	4145	0.26	7	354	0.08	2	3012	0.31	8	10498	0.24	8	(7) 1670647	0.13	469
DF	16	0.20	**0**	24	0.46	**0**	72	0.68	**0**	521	0.57	***2***	57	0.70	**0**	442	0.74	***2***	1214	0.78	3	171006	1.16	31
SHOP2	15	0.13	**0**	25	0.55	**0**	399	1.38	***1***	376	0.76	**1**	238	2.10	***1***	653	0.90	***2***	246446	1.47	46	115096	1.35	28
UMCP-BF	16	0.11	**0**	39	0.11	**0**	673	0.20	***1***	1080	0.17	4	2021	0.25	5	2165	0.26	5	112686	0.18	26	368516	0.11	79
UMCP-DF	15	0.12	**0**	20	0.33	**0**	389	1.38	**0**	339	0.87	**1**	352	1.79	***1***	273	0.62	**1**	238745	1.26	33	35242	1.62	***12***
UMCP-H	15	0.12	**0**	24	0.28	**0**	75	0.39	**0**	344	0.66	**1**	89	0.36	**0**	443	0.47	***2***	413	1.53	***1***	**6374**	0.79	**7**
$h_{\#M}$	18	0.17	**0**	26	0.39	**0**	157	0.25	***1***	968	0.43	5	136	0.10	***1***	784	0.46	5	2557	0.20	6	29040	1.26	15
$\lVert h_{\#M} \rVert$	***14***	0.08	**0**	**14**	0.00	**0**	42	0.11	**0**	620	0.62	3	43	0.30	***1***	325	0.76	***2***	1561	0.46	5	127033	1.52	31
$h_{\#F}$	17	0.16	**0**	26	0.40	**0**	111	0.25	**0**	793	0.30	4	86	0.16	***1***	620	0.43	3	1418	0.22	5	***18236***	0.74	***12***
$\lVert h_{\#F} \rVert$	***14***	0.09	**0**	***15***	0.08	**0**	72	0.28	**0**	450	0.29	***2***	33	0.14	**0**	252	0.59	**1**	3316	0.35	7	248677	0.84	45
$h_{\#F} + h_{TC+PC}$	16	0.20	**0**	24	0.39	**0**	***41***	0.08	**0**	880	0.34	5	54	0.36	**0**	373	0.74	***2***	940	0.38	4	60745	1.80	18
$\lVert h_{\#F} + h_{TC+PC} \rVert$	**13**	0.00	**0**	17	0.40	**0**	**22**	0.03	**0**	**92**	0.30	**1**	***31***	0.30	**0**	**123**	1.07	**1**	***46***	0.42	**0**	21939	1.29	***12***
$h_{\#F} + h_{MME}$	16	0.19	**0**	25	0.10	**0**	42	0.10	**0**	739	0.42	4	44	0.45	**0**	333	0.83	***2***	997	0.33	4	62353	1.77	18
$\lVert h_{\#F} + h_{MME} \rVert$	**13**	0.00	**0**	21	0.42	**0**	**22**	0.03	**0**	***105***	0.37	**1**	**27**	0.38	**0**	***160***	0.94	**1**	**37**	0.34	**0**	24459	1.30	***12***

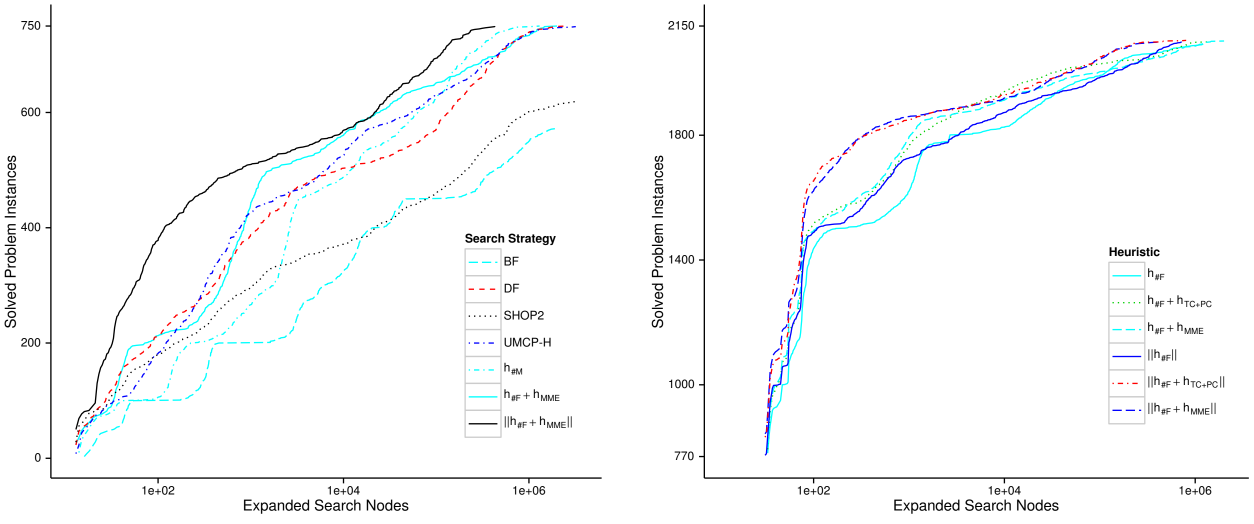
Problem	**3–1–3**			**3–2–1**			**3–2–2**			**3–2–3**			**3–3–1**			**3–3–2**			**3–3–3**		
Strategy	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	—	—	—	15174	0.15	13	710661	0.34	140	—	—	—	36408	0.14	17	322284	0.32	69	(16) 1122746	0.25	338
DF	668416	0.77	109	940	0.99	4	106748	1.22	28	318515	0.96	61	1639	0.95	5	43456	1.18	16	239418	0.89	48
SHOP2	472177	0.88	99	(40) 333398	2.41	64	(37) 294693	1.37	77	(11) 532783	1.29	133	(39) 82194	1.87	22	(29) 339167	1.22	83	(13) 461074	1.66	101
UMCP-BF	605241	0.10	122	810422	0.21	17	—	—	—	—	—	—	(42) 1628608	0.17	374	—	—	—	—	—	—
UMCP-DF	***129381***	0.89	**32**	(41) 113375	1.94	20	(48) 287016	1.52	63	762774	1.13	151	566905	1.72	80	(49) 350331	1.51	72	581613	0.68	133
UMCP-H	**125524**	0.77	***33***	1101	2.03	***2***	19992	1.03	***11***	(49) 636852	1.03	133	1206	1.36	***3***	24799	0.99	12	513303	0.58	122
$h_{\#M}$	219016	0.64	42	2208	0.14	9	36058	0.97	19	205674	0.63	46	2692	0.10	9	25465	0.97	15	247083	1.07	52
$\lVert h_{\#M} \rVert$	354407	0.91	72	1149	0.34	6	88179	0.96	29	308752	0.95	66	617	0.16	4	***5996***	0.99	***10***	184650	2.34	41
$h_{\#F}$	(42) 468944	0.73	101	1133	0.09	6	19827	0.48	17	122587	0.96	34	1162	0.14	5	30125	0.57	18	163326	0.93	37
$\lVert h_{\#F} \rVert$	245524	0.75	46	691	0.42	5	112265	1.12	31	126640	1.12	31	599	0.33	4	20535	2.22	13	64105	2.30	**19**
$h_{\#F} + h_{TC+PC}$	599905	0.47	106	410	0.37	3	**5174**	1.54	**10**	***84298***	1.88	**26**	729	0.35	4	**3972**	0.86	11	***50111***	1.93	**19**
$\lVert h_{\#F} + h_{TC+PC} \rVert$	132639	0.81	44	***242***	0.56	***2***	14283	1.63	12	113607	1.37	36	***474***	0.90	***3***	9438	1.68	***10***	**47144**	1.40	***20***
$h_{\#F} + h_{MME}$	807016	0.46	142	601	0.74	3	***11600***	1.42	***11***	179804	1.22	44	699	0.34	4	7477	2.02	**8**	85274	1.45	26
$\lVert h_{\#F} + h_{MME} \rVert$	145929	0.64	46	**128**	1.19	**1**	14862	2.46	***11***	**59059**	1.10	***27***	**248**	1.28	**2**	14435	2.03	***10***	51977	1.28	22

Table 3: Results for the **SmartPhone** domain.

Problem	**#1**			**#2**			**#3**		
Strategy	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
BF	30	0.14	**0**	486980	0.24	103	—	—	—
DF	20	0.06	**0**	(12) 166	1.57	***1***	164	1.04	**1**
SHOP2	20	0.08	**0**	(8) **82**	0.24	**0**	60486	2.30	22
UMCP-BF	58	0.24	**0**	—	—	—	375530	0.04	55
UMCP-DF	***19***	0.08	**0**	(5) 2033	1.27	2	15863	1.45	6
UMCP-H	**18**	0.00	**0**	—	—	—	15964	1.21	7
$h_{\#M}$	**18**	0.00	**0**	—	—	—	27114	0.04	17
$\lVert h_{\#M} \rVert$	**18**	0.00	**0**	—	—	—	27111	0.04	17
$h_{\#F}$	***19***	0.05	**0**	—	—	—	1608	0.61	9
$\lVert h_{\#F} \rVert$	20	0.00	**0**	—	—	—	826	0.78	7
$h_{\#F} + h_{TC+PC}$	22	0.08	**0**	—	—	—	***112***	0.40	***2***
$\lVert h_{\#F} + h_{TC+PC} \rVert$	21	0.02	**0**	(4) ***120***	0.11	***1***	**67**	0.15	**1**
$h_{\#F} + h_{MME}$	**18**	0.00	**0**	(2) 139	0.13	***1***	164	0.30	***2***
$\lVert h_{\#F} + h_{MME} \rVert$	20	0.07	**0**	—	—	—	230	0.89	***2***

Table 4: Results for the **Woodworking** domain.

#1			**#2**			**#3**			**#4**			**#5**		
μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t	μ_s	σ_s/μ_s	μ_t
23	0.02	**0**	***48***	0.01	**0**	60	0.05	**0**	565	0.25	**1**	474	0.27	***1***
23	0.02	**0**	**47**	0.01	**0**	46	0.18	**0**	210	0.41	**1**	168	0.43	***1***
30	0.59	**0**	96	1.90	***1***	332	1.56	***2***	(29) 198536	1.78	86	(30) 152786	1.93	59
26	0.18	**0**	54	0.19	**0**	74	0.15	**0**	1642	0.31	***2***	1612	0.18	2
24	0.11	**0**	***48***	0.07	**0**	51	0.25	**0**	164	0.32	**1**	167	0.34	***1***
25	0.11	**0**	49	0.10	**0**	50	0.25	**0**	115	0.34	**1**	106	0.39	**0**
23	0.02	**0**	**47**	0.01	**0**	***37***	0.07	**0**	99	0.17	**1**	74	0.19	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.07	**0**	103	0.24	**1**	71	0.23	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.02	**0**	***89***	0.20	**1**	***67***	0.12	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.02	**0**	90	0.27	**1**	**62**	0.10	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.03	**0**	92	0.24	**1**	71	0.18	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.03	**0**	101	0.51	**1**	72	0.27	***1***
23	0.02	**0**	**47**	0.01	**0**	**36**	0.02	**0**	**82**	0.47	**1**	84	0.43	***1***
23	0.02	**0**	**47**	0.01	**0**	***37***	0.07	**0**	171	0.47	**1**	175	0.37	***1***



(a) Satellite Domain

(b) All Domains

Figure 2: These plots show the number of solved problem instances (each run 50 times) over the number of expanded search nodes. For Fig. (a) only results for the Satellite domain were used, while Fig. (b) shows results over all domains. For the latter, we have omitted the data points below 750 solved instances for visual clarity.

We did not include it in the paper due to space restrictions, but we can report that the two plots look very similar indicating that the reduced search space pays off and comes with no additional overhead w.r.t. computation time.

SmartPhone The SmartPhone domain is the hardest domain that we evaluated. The first problem instance looks very similar to the UM-Translog instances: while our heuristics are among the best-performing strategies, there is only very little difference between the configurations. Even BF and DF find solutions very quickly. In the third problem instance, our strategies also perform very well. They expand at most 229 search nodes on average, where the best version of UMCP expands more than 150.000 nodes. Surprisingly, DF performs very well with only 163 expanded search nodes on average. The second problem instance seems to be the hardest one. All configurations except BF and DF produced timeouts or exceeded the memory limit in almost all runs. The SHOP2 configuration was able to find a solution very quickly (81 on average), but only in 8 out of 50 runs. Two of our proposed heuristics also solve the problem very quickly, but only in 2 and 4 runs, respectively.

Woodworking In all but one problems, $h_{\#F} + h_{MME}$ is among the two best-performing heuristics. However, clear conclusions cannot be drawn, since planning performance does not significantly vary among the deployed strategies.

Summary In summary, we have seen that our TDG-based heuristics need to expand the smallest number of search nodes in order to find a solution. To investigate the difference between the proposed heuristics, we include a plot (Fig. 2(b)) that shows the number of solved problem instances given a number of expanded search nodes for all problem instances among all domains.

We can draw several conclusions when investigating the data. First, we can see that the heuristics that need to explore the largest part of the search space are the relatively simple heuristics $h_{\#F}$ and $\|h_{\#F}\|$. This is our most important finding, since we can see that taking the TDG into account actually improves these heuristics significantly. It comes to our surprise, however, that there are no significant differences between the heuristics $\|h_{\#F} + h_{MME}\|$ and $\|h_{\#F} + h_{TC+PC}\|$. Another interesting result is that the normalized version of a heuristic clearly performs better on average than the corresponding non-normalized one.

Conclusion

We have proposed novel heuristics for Hybrid Planning that estimate the necessary number of modifications for a partial plan to turn it into a solution. These heuristics are based on a task decomposition graph and hence capable of incorporating the hierarchical aspects of the underlying domain. We conducted experiments using a novel algorithm for Hybrid Planning. Our heuristics proved to be the most informed ones in the evaluated problem instances.

Acknowledgment

We want to thank Thomas Geier and Bernd Schattenberg for proofreading the paper. This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Andrews, S.; Kettler, B.; Erol, K.; and Hendler, J. A. 1995. UM translog: A planning domain for the development and benchmarking of planning systems. Technical Report CS-TR-3487, Department of Computer Science, Institute of Systems Research, University of Maryland.

Bercher, P.; Geier, T.; Richter, F.; and Biundo, S. 2013. On delete relaxation in partial-order causal-link planning. In *Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, 674–681.

Bercher, P.; Geier, T.; and Biundo, S. 2013. Using state-based planning heuristics for partial-order causal-link planning. In *Advances in Artificial Intelligence, Proceedings of the 36nd German Conference on Artificial Intelligence (KI 2013)*, 1–12. Springer.

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief – a preliminary report on combining state abstraction and HTN planning. In *Proceedings of the 6th European Conference on Planning (ECP 2001)*, 157–168.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129:5–33.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving hierarchical planning performance by the use of landmarks. In *Proceedings of the 26th National Conference on Artificial Intelligence (AAAI 2012)*, 1763–1769. AAAI Press.

Elkawkagy, M.; Schattenberg, B.; and Biundo, S. 2010. Landmarks in hierarchical planning. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215, 229–234. IOS Press.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994)*, 249–254. AAAI Press.

Erol, K. 1996. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*. Ph.D. Dissertation, University of Maryland.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1955–1961. AAAI Press.

Joslin, D., and Pollack, M. E. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, 1004–1009. AAAI Press.

Kambhampati, S.; Mali, A.; and Srivastava, B. 1998. Hybrid planning for partially hierarchical domains. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI 1998)*, 882–888. AAAI Press.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 222–231. AAAI Press.

McAllester, D., and Rosenblitt, D. 1991. Systematic non-linear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI 1991)*, 634–639. AAAI Press.

Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, 459–466. Morgan Kaufmann.

Penberthy, J. S., and Weld, D. S. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the third International Conference on Knowledge Representation and Reasoning*, 103–114. Morgan Kaufmann.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP 2001)*, 37–48.

Russell, S., and Norvig, P. 1994. *Artificial Intelligence – A modern Approach*. Prentice-Hall, 1 edition.

Schattenberg, B.; Bidot, J.; and Biundo, S. 2007. On the construction and evaluation of flexible plan-refinement strategies. In *Advances in Artificial Intelligence, Proceedings of the 30th German Conference on Artificial Intelligence (KI 2007)*, 367–381. Springer.

Schattenberg, B.; Weigl, A.; and Biundo, S. 2005. Hybrid planning using flexible strategies. In *Advances in Artificial Intelligence, Proceedings of the 28th German Conference on Artificial Intelligence (KI 2005)*, 249–263. Springer.

Schattenberg, B. 2009. *Hybrid Planning & Scheduling*. Ph.D. Dissertation, University of Ulm, Germany.

Shivashankar, V.; Alford, R.; Kuter, U.; and Nau, D. 2013. The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2380–2386. AAAI Press.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research (JAIR)* 20:405–430.

The following pages show the publication:

P. Bercher, T. Geier, and S. Biundo. “Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning”. In: *Advances in Artificial Intelligence, Proceedings of the 36th German Conference on Artificial Intelligence (KI 2013)*. Springer, 2013, pp. 1–12. DOI: 10.1007/978-3-642-40942-4_1

https://link.springer.com/chapter/10.1007%2F978-3-642-40942-4_1

The final publication is available at link.springer.com

Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning

Pascal Bercher, Thomas Geier, and Susanne Biundo

Institute of Artificial Intelligence,
Ulm University, D-89069 Ulm, Germany,
email: firstName.lastName@uni-ulm.de

Abstract. We present a technique which allows partial-order causal-link (POCL) planning systems to use heuristics known from state-based planning to guide their search.

The technique encodes a given partially ordered partial plan as a new classical planning problem that yields the same set of solutions reachable from the given partial plan. As heuristic estimate of the given partial plan a state-based heuristic can be used estimating the goal distance of the initial state in the encoded problem. This technique also provides the first *admissible* heuristics for POCL planning, simply by using admissible heuristics from state-based planning. To show the potential of our technique, we conducted experiments where we compared two of the currently strongest heuristics from state-based planning with two of the currently best-informed heuristics from POCL planning.

1 Introduction

In most of today's classical planning approaches, problems are solved by informed (heuristic) progression search in the space of states. One reason for the success of this approach is the availability of highly informed heuristics performing a goal-distance estimate for a given state. In contrast, search nodes in plan-based search correspond to partially ordered partial plans; thus, the heuristics known from state-based planning are not directly applicable to plan-based search techniques.

One of the most important representatives of plan-based search is partial-order causal-link (POCL) planning [13, 17]. POCL planning benefits from its least-commitment principle enforcing decisions during planning only if necessary. For instance, POCL planning can be done *lifted* thereby avoiding premature variable bindings. POCL planning has greater flexibility at plan execution time [14] and eases the integration for handling resource or temporal constraints and durative actions [20, 3]. Its knowledge-rich plans furthermore enable the generation of formally sound plan explanations [19].

However, developing well-informed heuristics for POCL planning is a challenging task [21]; thus, heuristics are still rare. To address this shortcoming, we propose a technique which allows to use heuristics already known from state-based search in POCL planning, rather than developing *new* heuristics.

This technique works by transforming a current search node, i.e., a partially ordered partial plan, into a new planning problem, into which the given partial plan is completely encoded, s.t. solutions for the new problem correspond to solutions reachable from the encoded search node. Then, we evaluate the heuristic estimate of the transformed problem's initial state using any heuristic known from state-based search, and use it as heuristic estimate of the search node. As it turns out, not every state-based heuristic works with our technique, but we obtained promising empirical results for some of them.

The remainder of the paper is structured as follows: the next section is devoted to the problem formalization. Section 3 introduces the proposed transformation. In Section 4, we discuss several issues and questions arising when using the technique in practice. In Section 5, we evaluate our approach by comparing our POCL planning system using four different heuristics: two of them are the currently best-informed heuristics known for POCL planning, whereas the other two use state-of-the-art heuristics known from state-based planning in combination with our problem encoding. Finally, Section 6 concludes the paper.

2 POCL Planning

A planning domain is a tuple $\mathcal{D} = \langle \mathcal{V}, \mathcal{A} \rangle$, where $\mathcal{V}$ is a finite set of boolean *state variables*, $\mathcal{S} = 2^{\mathcal{V}}$ is the set of *states*, and $\mathcal{A}$ is a finite set of *actions*, each having the form $(\mathit{pre}, \mathit{add}, \mathit{del})$, where $\mathit{pre}, \mathit{add}, \mathit{del} \subseteq \mathcal{V}$. An action is applicable in a state $s \in \mathcal{S}$ if its precondition pre holds in s, i.e., $\mathit{pre} \subseteq s$. Its application generates the state $(s \setminus \mathit{del}) \cup \mathit{add}$. The applicability and application of action sequences is defined as usual. A *planning problem* in STRIPS notation [5] is a tuple $\pi = \langle \mathcal{D}, s_{\mathit{init}}, g \rangle$ with $s_{\mathit{init}} \in \mathcal{S}$ being the *initial state* and $g \subseteq \mathcal{V}$ being the *goal description.* A *solution* to π is an applicable action sequence starting in s_{init} and generating a state $s \supseteq g$ that satisfies the goal condition.

POCL planning is a technique that solves planning problems via search in the space of partial plans. A *partial plan* is a tuple $(\mathit{PS}, \prec, \mathit{CL})$. PS is a set of plan steps, each being a pair $l{:}a$ with an action $a \in \mathcal{A}$ and a unique label $l \in L$ with L being an infinite set of label symbols to differentiate multiple occurrences of the same action within a partial plan. The set $\prec \subset L \times L$ represents ordering constraints and induces a strict partial order on the plan steps in PS. CL is a set of causal links. A causal link $(l, v, l') \in L \times \mathcal{V} \times L$ testifies that the precondition $v \in \mathcal{V}$ of the plan step with label l' (called the *consumer*) is provided by the action with label l (called the *producer*). That is, if $l{:}(\mathit{pre}, \mathit{add}, \mathit{del}) \in \mathit{PS}$, $l'{:}(\mathit{pre}', \mathit{add}', \mathit{del}') \in \mathit{PS}$, and $(l, v, l') \in \mathit{CL}$, then $v \in \mathit{add}$ and $v \in \mathit{pre}'$. Furthermore, we demand $l \prec l'$ if $(l, v, l') \in \mathit{CL}$.

Now, π can be represented as a POCL planning problem $\langle \mathcal{D}, P_{\mathit{init}} \rangle$, where $P_{\mathit{init}} := (\{l_0{:}a_0, l_\infty{:}a_\infty\}, \{(l_0, l_\infty)\}, \emptyset)$ is the *initial partial plan.* The actions a_0 and a_∞ encode the initial state and goal description: a_0 has no precondition and s_{init} as add effect and a_∞ has g as precondition and no effects. A solution to such a problem is a partial plan P with no *flaws.* Flaws represent plan elements violating solution criteria. An *open precondition* flaw is a tuple $(v, l) \in \mathcal{V} \times L$

specifying that the precondition v of the plan step with label l is not yet protected by a causal link. A *causal threat* flaw is a tuple $(l,(l',v,l'')) \in L \times CL$ specifying that the plan step $l{:}(pre, add, del)$ with $v \in del$ may be ordered between the plan steps with label l' and l''. We say, the plan step with label l *threatens* the causal link (l',v,l''), since it might undo its protected condition v.

If a partial plan P has no flaws, every linearization of its plan steps respecting its ordering constraints is a solution to the planning problem in STRIPS notation. Hence, P is called a solution to the corresponding POCL problem.

POCL planning can be regarded as a refinement procedure [12], since it *refines* the initial partial plan P_{init} step-wise until a solution is generated. The algorithm works as follows [22]. First, a most-promising partial plan P is selected based on heuristics estimating the goal-distance or quality of P. Given such a partial plan P, a flaw selection function selects one of its flaws and resolves it. For that end, all *modifications* are applied, which are all possibilities to resolve the given flaw. A causal threat flaw $(l,(l',v,l'')) \in \mathcal{F}_{CausalThreat}$ can only be resolved by *promotion* or *demotion*. These modifications promote the plan step with label l before the one with label l', and demote it behind the one with label l'', respectively. An open precondition flaw $(v,l) \in \mathcal{F}_{OpenPrecondition}$ can only be resolved by inserting a causal link (l',v,l) which protects the open precondition v. This can be done either by using a plan step already present in the current partial plan, or by a new action from $\mathcal{A}$. The two-stage procedure of selecting a partial plan, calculating its flaws, and selecting and resolving a flaw is repeated until a partial plan P without flaws is generated. Hence, P is a solution to the POCL planning problem and returned.

3 Using State-Based Heuristics for POCL Planning

We encode a partially ordered partial plan into a new STRIPS planning problem. A similar encoding was already proposed by Ramírez and Geffner [18]. However, their encoding was used in the context of plan recognition for compiling observations away and it does not feature a partial order, causal links, nor did they state formal properties.

Given a planning problem in STRIPS notation $\pi = \langle\langle\mathcal{V},\mathcal{A}\rangle, s_{init}, g\rangle$ and a partial plan $P = (PS, \prec, CL)$, let $enc_{plan}(P,\pi) = \langle\langle\mathcal{V}',\mathcal{A}'\rangle, s'_{init}, g'\rangle$ be the *encoding* of P and π with:

$$\begin{aligned}
\mathcal{V}' &:= \mathcal{V} \cup \{l_-, l_+ \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}\\
\mathcal{A}' &:= \mathcal{A} \cup \{enc_{planStep}(l{:}a, \prec) \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}, \text{ with}\\
&\quad enc_{planStep}(l{:}(pre, add, del), \prec)\\
&\qquad := (pre \cup \{l_-\} \cup \{l'_+ \mid l' \prec l, l' \neq l_0\}, add \cup \{l_+\}, del \cup \{l_-\}),\\
s'_{init} &:= s_{init} \cup \{l_- \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}\\
g' &:= g \cup \{l_+ \mid l{:}a \in PS, l \notin \{l_0, l_\infty\}\}
\end{aligned}$$

The resulting problem subsumes the original one and extends it in the following way: all plan steps in P become additional actions in $\mathcal{A}'$ (we do not encode the artificial start and end actions, since their purpose is already reflected by the initial state and goal description). For each plan step $l{:}a$, we introduce two indicator variables l_- and l_+ that encode that $l{:}a$ has not or has been executed. Initially, none of the actions representing the encoding of these plan steps are marked as executed and the (additional) goal is to execute all of them. Furthermore, these actions use the indicator variables to ensure that they can only be executed in an order consistent with the partial order of the partial plan.

Although the encoding can be done in linear time [1], the effort for evaluating heuristics might increase as search progresses, since the resulting problem is of larger size than the original one. We will discuss this issue in the next section.

For the sake of simplicity, the formally specified function enc_{plan} ignores causal links. Since causal links induce additional constraints on a partial plan (cf. Section 2), compiling them away, too, captures even more information. The compilation can be done as follows: let (l_1, v, l_2) be a causal link, $l_1{:}a_1$ and $l_2{:}a_2$ the two corresponding plan steps, and a'_1 and a'_2 their encodings within $\mathcal{A}'$. We need to ensure that no action with v as delete effect can be inserted between a'_1 and a'_2. To that end, we introduce a counter variable $count(v)$ which counts how many causal links with the protected condition v are "currently active".[1] To update that counter correctly, any (encoded) action producing a causal link with condition v has the precondition $count(v) = i$ and the effect $count(v) = i + 1$. Analogously, any (encoded) action consuming such a causal link has the precondition $count(v) = i$ and the effect $count(v) = i - 1$. Then, any (encoded and non-encoded) action having v in its delete list has the precondition $count(v) = 0$. Note that the original planning problem does not need to be changed for every partial plan, although we need to add the precondition $count(v) = 0$ to each action for which there is a causal link (l_1, v, l_2) in the current partial plan. We can process the domain only *once* by adding the precondition $count(v) = 0$ to any action for any state variable v in advance. Concerning the overall runtime for compiling away causal links, assume a' being a (compiled) action consuming n and providing m causal links. Then, $|CL|^{(n+m)}$ actions must be created to provide all possible combinations of the *count* variables, where CL is the set of causal links of the partial plan to be encoded. The compilation is therefore exponential in the maximal number of preconditions and effects of all actions. Hence, assuming the planning *domain* is given in advance, our compilation is *polynomial.* In practice, it is also polynomial if the domain is not given in advance, because the maximal number of preconditions and effects is usually bounded by a small constant and does not grow with the domain description.

Before we can state the central property of the transformed problem, we need some further definitions. Let $P = (PS, \prec, CL)$ be a partial plan. Then, $ref(P) := \{\langle PS', \prec', CL' \rangle \mid PS' \supseteq PS, \prec' \supseteq \prec, CL' \supseteq CL\}$ is called the set of all *refinements* of P, i.e., the set of all partial plans which can be derived from P

[1] For the sake of simplicity, we use functions to describe the counter. Since these functions are simple increment and decrement operations, converting them into the STRIPS formalism is possible in linear time w.r.t. their maximum value which is bound by the number of causal links in the given partial plan.

by adding plan elements. Let $sol(\pi)$ be the set of all *solution* plans of π. We call $sol(\pi, P) := sol(\pi) \cap ref(P)$ the set of all solutions of π refining P.

Now, we define mappings to transform partial plans derived from the planning problem $enc_{plan}(P, \pi)$ to partial plans from the original planning problem π.[2] The functions $dec_{planStep}(l{:}(pre, add, del), \mathcal{V}) := l{:}(pre \cap \mathcal{V}, add \cap \mathcal{V}, del \cap \mathcal{V})$ and $dec_{plan}(\langle PS, \prec, CL\rangle, \pi) := \langle\{dec_{planStep}(l{:}a, \mathcal{V}) \mid l{:}a \in PS\}, \prec, \{(l, v, l') \in CL \mid v \in \mathcal{V}\}\rangle$ are called the *decoding* of a plan step and a partial plan, respectively. Thus, given a partial plan P' from the planning problem $enc_{plan}(P, \pi)$, $dec_{plan}(P', \pi)$ eliminates the additional variables and causal links used by the encoding.

The following proposition states that every solution of the original planning problem, which is also a refinement of the given partial plan, does also exist as a solution for the encoded problem and, furthermore, the converse holds as well: every solution of the encoded problem can be decoded into a solution of the original one, which is a refinement of the given partial plan, too. Thus, the set of solutions of the transformed planning problem is identical to the set of solutions of the original problem, reachable from the current partial plan.

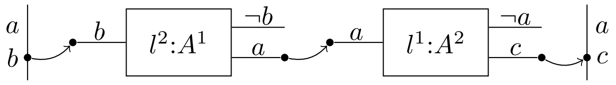
Proposition 1 *Let π be a planning problem and P a partial plan. It holds:*

- *if there is a partial plan P_{sol}, s.t. $P_{sol} \in sol(\pi, P)$, then there exists a partial plan P'_{sol} with $P'_{sol} \in sol(enc_{plan}(P, \pi))$ and $dec_{plan}(P'_{sol}, \pi) = P_{sol}$*
- *if there is a partial plan P'_{sol}, s.t. $P'_{sol} \in sol(enc_{plan}(P, \pi))$, then $dec_{plan}(P'_{sol}, \pi) \in sol(\pi, P)$*

Assuming the plan quality is based on action costs, we can use that proposition to find a heuristic function $h(\pi, P)$ that estimates the goal distance in π from the *partial plan* P by transforming π and P into the planning problem $\pi' = enc_{plan}(P, \pi)$ and setting $h(\pi, P) := \max\{h_{sb}(\pi', s'_{init}) - cost(P), 0\}$, where h_{sb} is any heuristic that takes a *state* as input. We need to subtract the action costs of P, since a heuristic estimate for P *excludes* the actions already present in P, whereas a heuristic estimate for s'_{init} should detect the necessity of inserting them and hence *includes* their cost as well. Taking the maximum of that value and zero is done in case the heuristic is overly positive and returns an estimated cost value lower than those of the actions already present in P.

Since, due to Proposition 1, the set of solutions of π is isomorphic to the solutions of π', even regarding action costs, using an admissible heuristic h_{sb} consequently makes $h(\pi, P)$ admissible, too. This is an interesting property of our approach, since there are no admissible POCL heuristics known to the literature.

Example. Let $\pi = \langle\langle\mathcal{V}, \mathcal{A}\rangle, s_{init}, g\rangle$ be a planning problem with $\mathcal{V} := \{a, b, c\}$, $\mathcal{A} := \{(\{b\}, \{a\}, \{b\}), (\{a\}, \{c\}, \{a\})\}$, $s_{init} := \{a, b\}$, and $g := \{a, c\}$. Let P be a partial plan which was obtained by a POCL algorithm as depicted below:



[2] For the sake of simplicity, our decoding assumes that no causal links were compiled away. Decoding the encoded causal links is straight-forward.

The arrows indicate causal links and A^1 and A^2 the actions of $\mathcal{A}$. P has only one open precondition: (a, l_∞), which encodes the last remaining goal condition.

The transformed problem, without compiling away causal links, is given by $enc_{\mathit{plan}}(P, \pi) = \langle\langle \mathcal{V}', \mathcal{A}'\rangle, s'_{init}, g'\rangle$ with:

$$\begin{aligned}
\mathcal{V}' &:= \{a, b, c, l^1_+, l^1_-, l^2_+, l^2_-\} \\
\mathcal{A}' &:= \{(\{b\}, \{a\}, \{b\}), && // \; A^1 \\
&\qquad (\{b, l^2_-\}, \{a, l^2_+\}, \{b, l^2_-\}), && // \; enc(l^2{:}A^1) \\
&\qquad (\{a\}, \{c\}, \{a\}), && // \; A^2 \\
&\qquad (\{a, l^1_-, l^2_+\}, \{c, l^1_+\}, \{a, l^1_-\})\} && // \; enc(l^1{:}A^2) \\
s'_{init} &:= \{a, b, l^1_-, l^2_-\} \\
g' &:= \{a, c, l^1_+, l^2_+\}
\end{aligned}$$

A heuristic estimate based on the transformed problem may incorporate the negative effects of $l^2{:}A^1$ and $l^1{:}A^2$ and has thus the potential to discover the partial plan/state to be invalid and thus prune the search space.

4 Discussion

Relaxation. Not every state-based heuristic is suited for our proposed approach. In order to determine how informed a chosen heuristic function is when used with our technique, one has to investigate the effect of the (heuristic-dependent) relaxation on the actions in $\mathcal{A}_{new} := \mathcal{A}' \setminus \mathcal{A}$. The actions in $\mathcal{A}_{new}$ (together with the additional goals) encode the planning progress so far, just like the current state does in state-based planning. Thus, relaxing them can have a strong impact on the resulting heuristic values. For instance, in our experiments, we noticed that the FF heuristic [10] always obtains the same estimates for the encoded problems of all search nodes making the heuristic blind.

Preprocessing. Some heuristics, like merge and shrink abstraction (M&S) [4, 9], perform a preprocessing step before the actual search and make up for it when retrieving each single heuristic value. Since we obtain a new planning problem for each single partial plan, a naive approach using this kind of heuristics would also perform that preprocessing in every search node, which is obviously not beneficial (and no *pre*-processing). Thus, given a specific heuristic, one has to investigate whether certain preprocessing steps can be done only *once* and then updated per partial plan if necessary.

Runtime. Although the transformation itself can be done efficiently, the time of evaluating heuristics might increase with the size of the encoded problem. At first glance, this might seem a strange property, since one would expect the heuristic calculation time either to remain constant (as for abstraction heuristics [4, 9]) or to *decrease* (as for the add or FF heuristics [6, 10]), as a partial plan comes closer to a solution. However, since partial plans are complex structures and many interesting decision problems involving them are NP hard w.r.t. their size [15], it is not surprising that evaluating heuristic estimates becomes more expensive as partial plans grow in size.

Ground Planning. The presentation of the proposed transformation in the paper assumes a ground problem representation. However, the approach also works for lifted planning without alterations to the encoding function. In lifted planning [22], the given partial plan is only *partially* ground, i.e., some action parameters are bound to constants, and the remaining ones are either unconstrained, codesignated or non-codesignated. Using the same encoding process but ignoring these designation constraints already works as described, since the initial state of the resulting encoded planning problem is still ground and evaluating its heuristic estimate is thus possible without alterations. Encoding the designation constraints is also possible, but ignoring them is just a problem relaxation as is ignoring causal links.

5 Evaluation

We implemented the described encoding without compiling away causal links in our POCL planning system. We compare the performance of planning using the encoding with two state-of-the-art state-based heuristics against the currently best-informed POCL heuristics. We also show results for a state-based planner.

The used POCL planner is implemented in Java®. As search strategy, we use weighted A* with weight 2. That is, a partial plan p with minimal f value is selected, given by $f(p) := g(p) + 2 * h(p)$ with $g(p)$ being the cost of p and $h(p)$ being its heuristic estimate. In cases where several partial plans have the same f value, we break ties by selecting a partial plan with higher cost; remaining ties are broken by the *LIFO* strategy thereby preferring the newest partial plan. As flaw selection function, we use a sequence of two flaw selection strategies. The first strategy prefers newest flaws (where all flaws detected in the same plan are regarded equally new). On a tie, we then use the *Least Cost Flaw Repair* strategy [11], which selects a flaw for which there are the least number of modifications, thereby minimizing the branching factor. Remaining ties are broken by chance. We configured our system to plan ground, because our current implementation only supports a ground problem encoding.

As POCL heuristics, we selected the two currently best-informed heuristics: the *Relax Heuristic* [16] and the *Additive Heuristic for POCL planning* [22]. They are adaptations of the *FF heuristic* [10] and the *add heuristic* [6], respectively. Both heuristics identify the open preconditions of the current partial plan and estimate the action costs to achieve them based on delete relaxation using a planning graph [2]. These heuristics are implemented natively in our system.[3]

[3] Our implementation of the Relax Heuristic takes into account *all* action costs in a relaxed plan, whereas the original version assumes cost 0 for all actions already present. We used our variant for the experiments, since it solved more problems than the original version.

As state-based heuristics, we chose the *LM-Cut* heuristic [8], which is a landmark-based heuristic and an admissible approximation to h^+, and the *Merge and Shrink* (M&S) heuristic [4, 9], which is an abstraction-based heuristic.

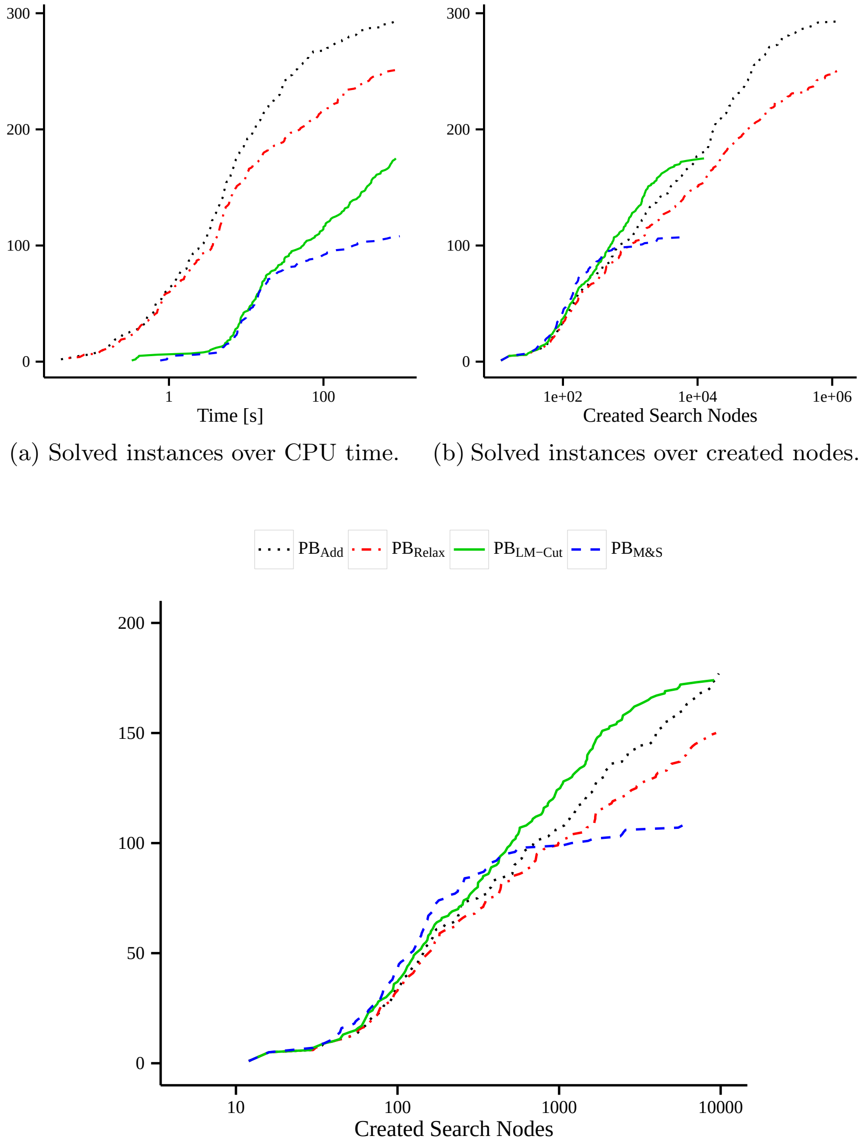
To evaluate heuristics from state-based planning, we chose to use an existing implementation. When planning using state-based heuristics, the POCL planner creates a domain and problem PDDL file for each search node encoding the corresponding partial plan, but ignoring causal links. We then use a modified version of the Fast Downward planning system [7] that exits after calculating the heuristic value for the initial state. While this approach saved us much implementation work, the obtained results are to be interpreted with care, since the process of calling another planning system in each search node is rather time-consuming: while the average runtime of the Add and Relax heuristics is at most 4 ms per search node over all evaluated problems, the LM-Cut heuristic has a mean runtime of 958 ms and a median of 225 ms. For the M&S heuristic[4], the mean is 7,500 ms and the median 542 ms. The very high runtimes of M&S are contributed to the fact that it performs a preprocessing step for every search node. Of course, in a native implementation of that heuristic in combination with our problem encoding, one would have to investigate whether an incremental preprocessing is possible as discussed in the last section.

Thus, the results using the state-based configurations are bound to be dominated by all others in terms of solved instances and runtime. Therefore, we focus our evaluation on the quality of the heuristics measured by the size of the produced search space in case a solution was found.

We evaluate on the non-temporal STRIPS problems taken from the International Planning Competitions (IPC) 1 to 5. Domains from the IPC 6 and 7 use action costs, which our system does not support. Missing domains from the IPC 1 to 5 use language features that cannot be handled either by our planner or by Fast Downward. From each domain we chose a number of instances consecutively, beginning with the smallest ones. The used domains and problems are given in Table 1; furthermore, the table contains the number of solved instances per domain by any of the four configurations. We also included results for the state-based planner Fast Downward. This planner, which is implemented in C++, clearly dominates our Java based POCL planner. For one problem, Fast Downward with M&S created 737 million nodes while our POCL planner created at most 2.9 million nodes, both for Add and Relax heuristic. Despite this discrepancy, the performance of the POCL planner using the Add heuristic surpasses Fast Downward using M&S in some domains.

The POCL planner was given 15 minutes of CPU time and 2GB of memory for each problem. For the configurations using the encoding, the call to Fast Downward also counts towards the time limit (including time spent generating the additional PDDL files), but not towards the memory limit. For the comparison with Fast Downward, we used a 15 minute wall time limit and no memory limit. All experiments were run on a 12 core Intel Xeon® with 2.4GHz.

[4] We chose f-preserving abstractions and 1500 abstract states. We chose a rather low number of abstract states to speed up the calculation time.



(a) Solved instances over CPU time.

(b) Solved instances over created nodes.

(c) Enlarged view of 1b; solved instances over created nodes.

Fig. 1: These plots show the number of solved instances on their y-axis, while the x-axis shows either CPU time or the number of created nodes. The configurations PB$_{\text{Add}}$ and PB$_{\text{Relax}}$ stand for POCL planning using the Add or the Relax heuristic, respectively. PB$_{\text{LM-Cut}}$ and PB$_{\text{M\&S}}$ denote POCL planning using the LM-Cut and the Merge and Shrink heuristic.

Figure 1a shows the number of solved instances over the used CPU time. As we can observe the transformation-based heuristics are severely dominated by the natively implemented POCL heuristics, as we expected. Figures 1b and 1c show the size of the explored search space over the number of solved instances. This means that configurations with higher curves solve more problems using the same number of visited nodes. We can observe that both transformation-based heuristics act more informed than the existing POCL heuristics in the beginning. The transformation-based heuristic using LM-Cut remains above the best existing POCL heuristic (Add) for the complete range of problems it could solve. When using M&S the performance deteriorates for the more complex problems, which we attribute to the small number of abstract states. In fact, a reasonable number of abstract states should be chosen domain dependently [9]. It is also the case that many runs experienced time outs after having explored only a small number of nodes. This means that the true curves of the transformation-based heuristics are expected to lie higher than depicted.

In summary, we can state that the experiments offer a promising perspective on the usefulness of the proposed transformation-based heuristics. In particular the LM-Cut heuristic proved to act more informed than the currently best known POCL heuristic, in addition to being the first known admissible one. Since the calculation of LM-Cut does not rely on preprocessing, like the Merge and Shrink heuristic does, we are optimistic that a native implementation of LM-Cut for POCL planning will offer a significant performance boost for POCL planning.

6 Conclusion

We presented a technique which allows planners performing search in the space of plans to use standard classical planning heuristics known from state-based planning. This technique is based on a transformation which encodes a given partial plan by means of an altered planning problem, s.t. evaluating the goal distance for the given partial plan corresponds to evaluating the goal distance for the initial state of the new planning problem.

We evaluated our approach by running our POCL planning system with two of the currently best-informed heuristics for POCL planning and two state-of-the-art-heuristics from state-based planning based on the proposed transformation. Whereas the first two heuristics are natively implemented in our system, the latter two are obtained by running Fast Downward in each search node and extracting its heuristic estimate for the initial state. The empirical data shows that our encoding works well with the evaluated state-based heuristics. In fact, one of these heuristics is even more informed than the best evaluated POCL heuristic, as it creates smaller search spaces in order to find solutions.

In future work we want to implement the encoding of causal links and evaluate our technique using a native implementation of the (state-based) LM-cut heuristic. Furthermore, we want to investigate whether the LM-cut heuristic can be *directly* transferred to the POCL setting without the compilation.

Table 1: This table gives the number of used problems per domain (n) and the number of solved instances per configuration and domain. The first configurations use our *plan*-based configurations and the right-most columns are the results of the *state-based* Fast Downward planner. All problems in the same block belong to the same IPC, from 1 to 5. A bold entry specifies the most number of solved instances among all configurations of the POCL planning system.

Domain	n	PB_{Add}	PB_{Relax}	$\text{PB}_{\text{LM-Cut}}$	$\text{PB}_{\text{M\&S}}$	$\text{SB}_{\text{LM-Cut}}$	$\text{SB}_{\text{M\&S}}$
grid	5	**0**	**0**	**0**	**0**	2	2
gripper	20	14	**20**	1	1	20	8
logistics	20	**16**	15	6	0	16	1
movie	30	**30**	**30**	**30**	**30**	30	30
mystery	20	8	**10**	5	5	13	13
mystery-prime	20	3	**4**	2	1	12	12
blocks	21	2	3	3	**5**	21	21
logistics	28	**28**	**28**	27	5	28	15
miconic	100	**100**	53	65	29	100	68
depot	22	**2**	**2**	1	1	11	7
driverlog	20	7	**9**	3	3	15	12
freecell	20	**0**	**0**	**0**	**0**	6	6
rover	20	**20**	19	9	5	18	8
zeno-travel	20	4	4	3	**5**	16	13
airport	20	**18**	15	6	5	20	18
pipesworld-noTankage	20	**8**	5	1	1	18	19
pipesworld-Tankage	20	**1**	**1**	**1**	**1**	11	14
satellite	20	**18**	**18**	4	3	15	7
pipesworld	20	**1**	**1**	**1**	**1**	11	14
rover	20	**0**	**0**	**0**	**0**	18	8
storage	20	7	**9**	5	5	17	15
tpp	20	**9**	8	5	5	9	7
total	526	296	254	178	111	427	318

Acknowledgements

This work is done within the Transregional Collaborative Research Centre SFB/ TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

1. Bercher, P., Biundo, S.: Encoding partial plans for heuristic search. In: Proceedings of the 4th Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2013). pp. 11–15 (2013)
2. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence 90, 281–300 (1997)
3. Coles, A., Coles, A., Fox, M., Long, D.: Forward-chaining partial-order planning. In: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010). pp. 42–49. AAAI Press (2010)

4. Dräger, K., Finkbeiner, B., Podelski, A.: Directed model checking with distance-preserving abstractions. In: Valmari, A. (ed.) SPIN. Lecture Notes in Computer Science, vol. 3925, pp. 19–34. Springer (2006)
5. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208 (1971)
6. Haslum, P., Geffner, H.: Admissible heuristics for optimal planning. In: Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000). pp. 140–149. AAAI Press (2000)
7. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research (JAIR) 26, 191–246 (2006)
8. Helmert, M., Domshlak, C.: Landmarks, critical paths and abstractions: Whats the difference anyway? In: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009). vol. 9, pp. 162–169 (2009)
9. Helmert, M., Haslum, P., Hoffmann, J.: Flexible abstraction heuristics for optimal sequential planning. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). pp. 176–183 (2007)
10. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research (JAIR) 14, 253–302 (May 2001)
11. Joslin, D., Pollack, M.E.: Least-cost flaw repair: A plan refinement strategy for partial-order planning. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994). pp. 1004–1009. AAAI Press (1994)
12. Kambhampati, S.: Refinement planning as a unifying framework for plan synthesis. AI Magazine 18(2), 67–98 (1997)
13. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI 1991). pp. 634–639. AAAI Press (1991)
14. Muise, C., McIlraith, S.A., Beck, J.C.: Monitoring the execution of partial-order plans via regression. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 1975–1982. AAAI Press (2011)
15. Nebel, B., Bäckström, C.: On the computational complexity of temporal projection, planning, and plan validation. Artificial Intelligence 66(1), 125–160 (1994)
16. Nguyen, X., Kambhampati, S.: Reviving partial order planning. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001). pp. 459–466. Morgan Kaufmann (2001)
17. Penberthy, J.S., Weld, D.S.: UCPOP: A sound, complete, partial order planner for ADL. In: Proceedings of the third International Conference on Knowledge Representation and Reasoning. pp. 103–114. Morgan Kaufmann (1992)
18. Ramírez, M., Geffner, H.: Plan recognition as planning. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). pp. 1778–1783. AAAI Press (July 2009)
19. Seegebarth, B., Müller, F., Schattenberg, B., Biundo, S.: Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In: Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012). pp. 225–233. AAAI Press (6 2012)
20. Vidal, V., Geffner, H.: Branching and pruning: An optimal temporal POCL planner based on constraint programming. Artificial Intelligence 170(3), 298–335 (2006)
21. Weld, D.S.: Systematic nonlinear planning: A commentary. AI Magazine 32(1), 101–103 (2011)
22. Younes, H.L.S., Simmons, R.G.: VHPOP: Versatile heuristic partial order planner. Journal of Artificial Intelligence Research (JAIR) 20, 405–430 (2003)

The following pages show the publication:

M. Elkawkagy, P. Bercher, B. Schattenberg, and S. Biundo. “Improving Hierarchical Planning Performance by the Use of Landmarks”. In: *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012).* AAAI Press, 2012, pp. 1763–1769

Reprinted with kind permission of AAAI Press.

Improving Hierarchical Planning Performance by the Use of Landmarks

Mohamed Elkawkagy and **Pascal Bercher** and **Bernd Schattenberg** and **Susanne Biundo**
Institute of Artificial Intelligence,
Ulm University, D-89069 Ulm, Germany,
email: *forename.surname*@uni-ulm.de

Abstract

In hierarchical planning, landmarks are tasks that occur on every search path leading from the initial plan to a solution. In this work, we present novel domain-independent planning strategies based on such hierarchical landmarks. Our empirical evaluation on four benchmark domains shows that these landmark-aware strategies outperform established search strategies in many cases.

1 Introduction

While landmarks are widely used to improve the performance of classical planners, a different notion of landmarks has recently been developed for HTN-based approaches (Elkawkagy, Schattenberg, and Biundo 2010). Unlike the classical case where landmarks are facts that must hold in some intermediate state of any solution plan, hierarchical landmarks are mandatory tasks – tasks that have to be decomposed on any search path leading from the initial plan to a solution of the planning problem.

Hierarchical task network (HTN) planning relies on the concepts of tasks and methods (Erol, Hendler, and Nau 1994). While primitive tasks correspond to classical planning operators, abstract tasks are a means to represent complex activities. For each abstract task, a number of (decomposition) methods are available, each of which provides a task network, i.e., a plan that specifies a predefined (abstract) solution of the task. Planning problems are (initial) task networks; they are solved by incrementally decomposing the abstract tasks until the network contains only executable primitive ones.

Strategies of HTN-based planners differ in the ways they select appropriate methods and interleave the decomposition of tasks with measures to resolve causal interactions between tasks. Systems of the SHOP family, like SHOP2, expand tasks in the order in which they are to be executed and consider causality only on primitive levels (Nau et al. 2003). Other strategies alternate task decomposition and causal conflict resolution (McCluskey 2000) or comply with the current state of the task network (Schattenberg, Bidot, and Biundo 2007).

In this paper, we describe how the exploitation of landmark information leads to novel domain-independent search strategies for HTN-based planning. A so-called landmark table is extracted from the current planning problem in a pre-processing step. It lists the landmark tasks and reveals the various options at hand. Options are tasks that are not mandatory, but may have to be decomposed depending on the method that is selected to implement the respective landmark. This information is used to compute the expansion effort of the problem – a heuristic to guide the selection of methods and with that reduce the effective branching factor of the search space.

We implemented the landmark-aware planning strategies in our experimental setting and evaluated their performance on four different benchmark domains. It turned out that our novel strategies outperform their conventional counterparts on practically all problems if the decomposition hierarchy of the underlying domain is of non-trivial depth.

In classical state-based planning the concept of landmarks (Porteous, Sebastia, and Hoffmann 2001) enabled the development of strong heuristics (Helmert and Domshlak 2009; Bonet and Helmert 2010). One of the currently best performing planners uses such a landmark heuristic (Richter and Westphal 2010). The work of Zhu and Givan (2004) generalized landmarks to so-called action landmarks. Marthi, Russell, and Wolfe (2008) specify a semantics for HTN planning in which the preconditions and effects of abstract tasks can be interpreted as abstract landmarks, as they are gained via incremental abstraction of more basic tasks.

In the following, we briefly introduce the underlying planning framework and the concept of hierarchical landmarks. We then define the landmark-aware strategies and describe the experimental setting as well as the evaluation results.

2 Planning Framework

The planning framework is based on a formalization which is the fusion of HTN planning with partial-order causal-link (POCL) planning (Biundo and Schattenberg 2001). A *task* (or *task schema*) $t(\overline{\tau}) = \langle \text{prec}, \text{eff} \rangle$ specifies preconditions and effects via conjunctions of literals over the task parameters $\overline{\tau} = \tau_1 \ldots \tau_n$. *States* are sets of literals. Applicability of tasks and the state transformations caused by their execution are defined as usual. A (partial) *plan* $P = \langle S, \prec, V, C \rangle$ consists of a set S of *plan steps*, i.e., uniquely labeled task instances, a set $\prec$ of *ordering constraints* that impose a partial order on S, a set V of *variable constraints*, and a set C of *causal links*. V consists of (in)equations that associate task parameters with other parameters or constants. We denote by $Ground(S, V)$ the set of ground tasks obtained by replacing all parameters of all tasks in P with constants in a way compatible with V. The causal links are adopted from POCL planning: A causal link $l{:}t(\overline{\tau}) \rightarrow_{\varphi} l'{:}t'(\overline{\tau}')$ indicates that φ is implied by the precondition of plan step $l'{:}t'(\overline{\tau}')$ and at the same time is a consequence of the effects of plan step $l{:}t(\overline{\tau})$. Hence, φ is said to be *supported* this way. A task is called *abstract* if at least one method is provided for refining it, otherwise it is called *primitive*. A method $m = \langle t(\overline{\tau}), P\rangle$ relates the abstract task $t(\overline{\tau})$ to the plan P, which is called an *implementation* of $t(\overline{\tau})$.

An HTN planning problem $\Pi = \langle D, s_{\text{init}}, P_{\text{init}}\rangle$ is composed of a domain model $D = \langle T, M\rangle$, where T and M denote finite sets of tasks and methods, an initial state s_{init}, and an initial plan P_{init}. A plan $P = \langle S, \prec, V, C\rangle$ is a solution to Π if and only if: (1) P is a refinement of P_{init}, i.e., a successor of the initial plan in the induced search space (see Def. 1 below); (2) each precondition of a plan step in S is supported by a causal link in C and no such link is threatened, i.e., for each $l{:}t(\overline{\tau}) \rightarrow_{\varphi} l'{:}t'(\overline{\tau}')$ the ordering constraints in $\prec$ ensure that no plan step $l''{:}t''(\overline{\tau}'')$ with an effect that is unifiable with $\neg\varphi$ can be placed between $l{:}t(\overline{\tau})$ and $l'{:}t'(\overline{\tau}')$; (3) the ordering constraints are consistent, i.e., $\prec$ respects the ordering implied by C and it does not induce cycles on S; (4) the variable constraints are consistent, i.e., the (in)equations in V are not contradictory; and (5) all plan steps in S correspond to primitive ground tasks.

$\mathcal{S}ol_{\Pi}$ denotes the set of all solutions of Π.

Please note that we encode the initial state via the effects of an artificial primitive "start" task, as it is usually done in POCL planning. In doing so, criteria (2) to (5) guarantee that the solution is executable in the initial state.

In order to refine the initial plan into a solution, there are various *refinement steps* (or *plan modifications*) available; in HTN planning, these are: (1) The decomposition of abstract tasks using methods, (2) the insertion of causal links to support open preconditions of plan steps, (3) the insertion of ordering constraints, and (4) the insertion of variable constraints. Given an HTN planning problem we can define the induced search space as follows.

Definition 1 (Induced Search Space) *The directed graph $\mathcal{P}_{\Pi} = \langle \mathcal{V}_{\Pi}, \mathcal{E}_{\Pi}\rangle$ with vertices $\mathcal{V}_{\Pi}$ and edges $\mathcal{E}_{\Pi}$ is called the* induced search space of the planning problem Π *if and only if (1) $P_{\text{init}} \in \mathcal{V}_{\Pi}$, (2) if there is a plan modification refining $P \in \mathcal{V}_{\Pi}$ into a plan P', then $P' \in \mathcal{V}_{\Pi}$ and $(P, P') \in \mathcal{E}_{\Pi}$, and (3) $\mathcal{P}_{\Pi}$ is minimal such that (1) and (2) hold.*

For $\mathcal{P}_{\Pi} = \langle \mathcal{V}_{\Pi}, \mathcal{E}_{\Pi}\rangle$, we write $P \in \mathcal{P}_{\Pi}$ instead of $P \in \mathcal{V}_{\Pi}$. Note that $\mathcal{P}_{\Pi}$ is in general neither acyclic nor finite. For the former, consider a planning problem in which there are the abstract tasks $t(\overline{\tau})$, $t'(\overline{\tau}')$ as well as two methods, each of which transforms one task into the other. For the latter, consider a planning problem containing an abstract task $t(\overline{\tau})$ and a primitive task $t'(\overline{\tau}')$ as well as two methods for $t(\overline{\tau})$: one maps $t(\overline{\tau})$ to a plan containing only $t'(\overline{\tau}')$, the other maps $t(\overline{\tau})$ to a plan containing $t'(\overline{\tau}')$ and $t(\overline{\tau})$ thus enabling the construction of arbitrary long plans.

In order to search for solutions the induced search space is explored in a heuristically guided manner by the following generic refinement planning algorithm:

Algorithm 1: Refinement Planning Algorithm

```
Input  : The sequence $\texttt{Fringe} = \langle P_{\text{init}}\rangle$.
Output : A solution or fail.
1 while $\texttt{Fringe} = \langle P_1 \ldots P_n\rangle \neq \varepsilon$ do
2     $F \leftarrow f^{\text{FlawDet}}(P_1)$
3     if $F = \emptyset$ then return $P_1$
4     $\langle \texttt{m}_1 \ldots \texttt{m}_k\rangle \leftarrow f^{\text{ModOrd}}(\bigcup_{\texttt{f} \in F} f^{\text{ModGen}}(\texttt{f}))$
5     $\texttt{succ} \leftarrow \langle \texttt{app}(\texttt{m}_1, P_1) \ldots \texttt{app}(\texttt{m}_k, P_1)\rangle$
6     $\texttt{Fringe} \leftarrow f^{\text{PlanOrd}}(\texttt{succ} \circ \langle P_2 \ldots P_n\rangle)$
7 return fail
```

The fringe $\langle P_1 \ldots P_n\rangle$ is a sequence containing all unexplored plans that are direct successors of visited non-solution plans in $\mathcal{P}_{\Pi}$. It is ordered in a way such that a plan P_i is estimated to lead more quickly to a solution than plans P_j for $j > i$. The current plan is always the first plan of the fringe. The planning algorithm iterates on the fringe as long as no solution is found and there are still plans to refine (line 1). Hence, the flaw detection function f^{FlawDet} in line 2 calculates all flaws of the current plan. A flaw is a set of plan components that are involved in the violation of a solution criterion. The presence of an abstract task raises a flaw that consists of that task, a causal threat consists of a causal link and the threatening plan step, for example. If no flaws can be found, the plan is a solution and returned (line 3). In line 4, the modification generating function f^{ModGen} calculates all plan modifications that address the flaws of the current plan. Afterwards, the modification ordering function f^{ModOrd} orders these modifications according to a given strategy. The fringe is finally updated in two steps: First, the plans resulting from applying the modifications are computed (line 5) and put at the beginning of the fringe (line 6). Second, the plan ordering function f^{PlanOrd} orders the updated fringe. This step can also be used to discard plans, i.e., to delete plans permanently from the fringe. This is useful for plans that contain unresolvable flaws like an inconsistent ordering of tasks. If the fringe becomes empty, no solution exists and `fail` is returned.

In this setting, the search strategy appears as a combination of the plan modification and plan ordering functions. In order to perform a depth first search, for example, the plan ordering is the identity function ($f^{\text{PlanOrd}}(\overline{P}) = \overline{P}$ for any sequence $\overline{P}$), whereas the modification ordering f^{ModOrd} determines which branch of the search space to visit first.

3 Landmarks

The landmark-aware planning strategies rely on hierarchical and local landmarks – ground tasks that occur in the plan sequences leading from a problem's initial plan to its solution.

Definition 2 (Solution Sequences) *Let $\langle \mathcal{V}_{\Pi}, \mathcal{E}_{\Pi}\rangle$ be the induced search space of the planning problem Π. Then, for* *any plan $P \in \mathcal{V}_\Pi$, $\mathcal{S}ol\mathcal{S}eq_\Pi(P) := \{\langle P_1 \ldots P_n\rangle \mid P_1 = P$, $P_n \in \mathcal{S}ol_\Pi$, and for all $1 \leq i < n$, $(P_i, P_{i+1}) \in \mathcal{E}_\Pi\}$.*

Definition 3 (Landmark) *A ground task $t(\overline{\tau})$ is called a* landmark *of planning problem Π, if and only if for each $\langle P_1 \ldots P_n\rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P_{\text{init}})$ there is an $1 \leq i \leq n$, such that $t(\overline{\tau}) \in Ground(S_i, V_n)$ for $P_i = \langle S_i, \prec_i, V_i, C_i\rangle$ and $P_n = \langle S_n, \prec_n, V_n, C_n\rangle$.*

While a landmark occurs in every plan sequence that is rooted in the initial plan and leads towards a solution, a *local* landmark occurs merely in each such sequence rooted in a plan containing a specific abstract ground task $t(\overline{\tau})$.

Definition 4 (Local Landmark of an Abstract Task) *For an abstract ground task $t(\overline{\tau})$ let $\mathcal{P}_\Pi(t(\overline{\tau})) := \{P \in \mathcal{P}_\Pi \mid P = \langle S, \prec, V, C\rangle$ and $t(\overline{\tau}) \in Ground(S, V)\}$.*
A ground task $t'(\overline{\tau}')$ is a local landmark *of $t(\overline{\tau})$, if and only if for all $P \in \mathcal{P}_\Pi(t(\overline{\tau}))$ and each $\langle P_1 \ldots P_n\rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P)$ there is an $1 \leq i \leq n$, such that $t'(\overline{\tau}') \in Ground(S_i, V_n)$ for $P_i = \langle S_i, \prec_i, V_i, C_i\rangle$ and $P_n = \langle S_n, \prec_n, V_n, C_n\rangle$.*

Since there are only finitely many tasks and we assume only finitely many constants, there is only a finite number of (local) landmarks.

Given a planning problem Π, the relevant landmark information can be extracted in a pre-processing step. We use the extraction procedure introduced in previous work of the authors (Elkawkagy, Schattenberg, and Biundo 2010) and assume that the information is already stored in a so-called *landmark table*. Its definition relies on a task decomposition graph, which is a relaxed representation of how the initial plan of a planning problem can be decomposed.

Definition 5 (Task Decomposition Graph) *The directed bipartite graph $\langle V_T, V_M, E\rangle$ with task vertices V_T, method vertices V_M, and edges E is called the task decomposition graph (TDG) of the planning problem Π if and only if*

1. *$t(\overline{\tau}) \in V_T$ for all $t(\overline{\tau}) \in Ground(S, V)$, for $P_{init} = \langle S, \prec, V, C\rangle$,*
2. *if $t(\overline{\tau}) \in V_T$ and if $\langle t(\overline{\tau}'), \langle S, \prec, V, C\rangle\,\rangle \in M$ s.t. $\overline{\tau}$ is compatible with $\overline{\tau}'$ and V, then*
   - (a) *$\langle t(\overline{\tau}), \langle S, \prec, V', C\rangle\rangle \in V_M$ such that $V' \supseteq V$ binds all variables in S to a constant and*
   - (b) *$(t(\overline{\tau}), \langle t(\overline{\tau}), \langle S, \prec, V', C\rangle\rangle) \in E$,*
3. *if $\langle t(\overline{\tau}), \langle S, \prec, V, C\rangle\,\rangle \in V_M$, then*
   - (a) *$t'(\overline{\tau}') \in V_T$ for all $t'(\overline{\tau}') \in Ground(S, V)$ and*
   - (b) *$(\langle t(\overline{\tau}), \langle S, \prec, V, C\rangle\,\rangle, t'(\overline{\tau}')) \in E$, and*
4. *$\langle V_T, V_M, E\rangle$ is minimal such that (1), (2), and (3) hold.*

Note that the TDG of a planning problem is always finite as there are only finitely many methods and ground tasks.

Please also note that, due to the uninformed instantiation of unbound variables in a decomposition step in criterion 2.(a), the TDG of a planning problem generally becomes intractably large. We hence prune parts of the TDG which can provably be ignored due to a relaxed reachability analysis of primitive tasks. This pruning technique is described in our earlier work (Elkawkagy, Schattenberg, and Biundo 2010).

The *landmark table* represents a (possibly pruned) TDG plus additional information about local landmarks.

Definition 6 (Landmark Table) *Let $\langle V_T, V_M, E\rangle$ be a (possibly pruned) TDG of the planning problem Π. The* landmark table *of Π is the set $LT = \{\langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle \mid t(\overline{\tau}) \in V_T$ abstract ground task$\}$, where $M(t(\overline{\tau}))$ and $O(t(\overline{\tau}))$ are defined as follows:*

$$M(t(\overline{\tau})) := \{t'(\overline{\tau}') \in V_T \mid t'(\overline{\tau}') \in Ground(S, V) \text{ for all } \langle t(\overline{\tau}), \langle S, \prec, V, C\rangle\,\rangle \in V_M\}$$

$$O(t(\overline{\tau})) := \{\,Ground(S, V) \setminus M(t(\overline{\tau})) \mid \langle t(\overline{\tau}), \langle S, \prec, V, C\rangle\,\rangle \in V_M\}$$

Each landmark table entry partitions the tasks introduced by decompositions into two sets: Mandatory tasks $M(t(\overline{\tau}))$ are those ground tasks that are contained in all plans introduced by some method which decomposes $t(\overline{\tau})$; hence, they are local landmarks of $t(\overline{\tau})$. The optional task set $O(t(\overline{\tau}))$ contains for each method decomposing $t(\overline{\tau})$ the set of ground tasks which are not in the mandatory set; it is hence a set of sets of tasks.

Please note that the landmark table encodes a possibly pruned TDG and is thus not unique. In fact, various local landmarks might only be detected after pruning. For instance, suppose an abstract task has three available methods, two of which have some tasks in their referenced plans in common. However, the plan referenced by the third method is disjoint to the other two. Hence, the mandatory sets are empty. If the third method can be proven to be infeasible and is hence pruned from the TDG, the mandatory set will contain those tasks the plans referenced by the first two methods have in common.

Example

The following example will demonstrate how the TDG and a landmark table of a planning problem looks like.

Let $\Pi = \langle D, s_{\text{init}}, P_{\text{init}}\rangle$ an HTN planning problem with $D = \langle\{t_1(\tau_1), \ldots, t_5(\tau_5)\}, \{m_a, m'_a, m_b, m'_b\}\rangle$, $P_{\text{init}} = \langle\{l_1{:}t_1(\tau_1)\}, \{\tau_1{=}c_1\}\rangle$[1], and constants c_1 and c_2, where:

$$m_a := \langle t_1(\tau_1), \langle\{l_1{:}t_3(\tau_1), l_2{:}t_3(\tau_2), l_3{:}t_2(\tau_1)\}, \{\tau_1{\neq}\tau_2\}\rangle\rangle$$
$$m'_a := \langle t_1(\tau_1), \langle\{l_4{:}t_2(\tau_1), l_5{:}t_1(\tau_1)\}, \emptyset\rangle\rangle$$
$$m_b := \langle t_3(\tau_1), \langle\{l_6{:}t_4(\tau_1), l_7{:}t_5(\tau_1)\}, \emptyset\rangle\rangle$$
$$m'_b := \langle t_3(\tau_1), \langle\{l_8{:}t_4(\tau_1)\}, \emptyset\rangle\rangle$$

The TDG for Π is given in Figure 1; the according landmark table is given as follows:

Abs. Task	Mandatory	Optional
$t_1(c_1)$	$\{t_2(c_1)\}$	$\{\{t_3(c_2), t_3(c_1)\}, \{t_1(c_1)\}\}$
$t_3(c_2)$	$\{t_4(c_2)\}$	$\{\emptyset, \{t_5(c_2), t_2(c_2)\}\}$
$t_3(c_1)$	$\{t_4(c_1)\}$	$\{\emptyset, \{t_5(c_1), t_2(c_1)\}\}$

4 Landmark-Aware Strategies

Exploiting landmarks during planning is based on the idea of treating landmarks as characteristic, “inevitable” elements on the refinement paths to any solution. The mandatory sets

[1] As our example comes without ordering constraints and causal links, we give plans as 2-tuples $P = \langle S, V\rangle$.

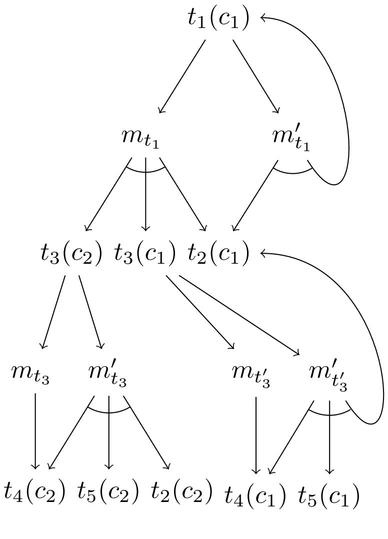


Figure 1: The TDG for the planning problem Π. The method vertices are given as follows:
$m_{t_1} = \langle t_1(c_1), m_a|_{\tau_1=c_1,\tau_2=c_2}\rangle$, $m'_{t_1} = \langle t_1(c_1), m'_a|_{\tau_1=c_1}\rangle$,
$m_{t_3} = \langle t_3(c_2), m_b|_{\tau_1=c_2}\rangle$, $m'_{t_3} = \langle t_3(c_2), m'_b|_{\tau_1=c_2}\rangle$,
$m_{t'_3} = \langle t_3(c_1), m_b|_{\tau_1=c_1}\rangle$, $m'_{t'_3} = \langle t_3(c_1), m'_b|_{\tau_1=c_2}\rangle$

in the landmark table do not contribute directly to the identification of a solution path. They do, however, allow to estimate upper and lower bounds for the number of expansions an abstract task requires before a solution is found: A landmark table entry $\langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle$ denotes that all tasks in $M(t(\overline{\tau}))$ are introduced into the refinement plan, no matter which method is used for decomposing $t(\overline{\tau})$. With the optional tasks at hand we can now infer that in the most optimistic case a solution can be developed straight from the implementation of the method with the "smallest" remains according to $O(t(\overline{\tau}))$. Following a similar argument, adding the efforts for all implementations stored in $O(t(\overline{\tau}))$ allows to estimate an upper bound for the "expansion effort".

From the above considerations, two essential properties of our landmark-aware strategies emerge: First, since the landmark exploitation will be defined in terms of measuring expansion alternatives, the resulting strategy component has to be a modification ordering function. Second, if we base the modification preference on the optional sets in the landmark table entries, we implement an abstract view on the method definition that realizes the least-commitment principle.

Concerning the first two strategies below, we interpret the term "expansion effort" literally and therefore define "smallest" method to be the one with the fewest abstract tasks in the implementing plan. To this end, we define the cardinality of a set of tasks in terms of the number of corresponding entries that a given landmark table does contain.

Definition 7 (Landmark Cardinality) *Given a landmark table LT, we define the* landmark cardinality *of a set of tasks $o = \{t_1(\overline{\tau}_1), \ldots, t_n(\overline{\tau}_n)\}$ to be*

$$|o|_{LT} := |\{t(\overline{\tau}) \in o \mid \langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle \in LT\}|$$

A heuristic based on this information can heavily overestimate the search effort because the landmark table typically contains a number of tasks that turn out to be unachievable in the given problem. The strategy also does not take into account the refinement effort it takes to make an implementation operational on the primitive level by establishing causal links, resolving causal threats, and grounding tasks. For the time being, we assume that all methods deviate from a perfect heuristic estimate more or less to the same amount. We will see that this simplification actually yields a heuristic with good performance.

Definition 8 (Landmark-aware strategy lm_1) *Given a plan $P = \langle S, \prec, V, C\rangle$, let $t_i(\overline{\tau}_i)$ and $t_j(\overline{\tau}_j)$ be ground instances of two abstract tasks in S that are compatible with the (in)equations in V and that are referenced by two abstract task flaws $\mathtt{f}_i$ and $\mathtt{f}_j$, respectively, that are found in P. Let a given landmark table LT contain the corresponding entries $\langle t_i(\overline{\tau}_i), M(t_i(\overline{\tau}_i)), O(t_i(\overline{\tau}_i))\rangle$ and $\langle t_j(\overline{\tau}_j), M(t_j(\overline{\tau}_j)), O(t_j(\overline{\tau}_j))\rangle$.*

The modification ordering function lm_1 orders a plan modification $\mathtt{m}_i$ before $\mathtt{m}_j$ if and only if $\mathtt{m}_i$ addresses $\mathtt{f}_i$, $\mathtt{m}_j$ addresses $\mathtt{f}_j$, and

$$\sum_{o \in O(t_i(\overline{\tau}_i))} |o|_{LT} < \sum_{o \in O(t_j(\overline{\tau}_j))} |o|_{LT}$$

This strategy implements the least commitment principle, as it favors those decomposition plan refinements that impose fewer successor plans. It reduces the effective branching factor of the search space (cf. *fewest alternatives first* heuristic in HTN planning (Tsuneto, Nau, and Hendler 1997)). The proper choice of the ground task instances $t_i(\overline{\tau}_i)$ and $t_j(\overline{\tau}_j)$ in the above definition is crucial for the actual performance, however, because the plan modifications typically operate on the lifted abstract tasks and method definitions.

While the above heuristic focuses on the very next level of refinement, a strategy should also take estimates for subsequent refinement levels into account, thus minimizing the number of refinement choices until no more decompositions are necessary. To this end, for a given landmark table LT, let $O^*(t(\overline{\tau}))$ be the transitive closure of the optional sets on a recursive traversal of the table entries, beginning in $t(\overline{\tau})$.

Definition 9 (Closure of the Optional Set) *The* closure of the optional set *for a given ground task $t(\overline{\tau})$ and a landmark table LT is the smallest set $O^*(t(\overline{\tau}))$, such that $O^*(t(\overline{\tau})) = \emptyset$ for primitive $t(\overline{\tau})$, and otherwise:*

$$O^*(t(\overline{\tau})) = O(t(\overline{\tau})) \cup \bigcup_{o \in O(t(\overline{\tau}))} \Big(\bigcup_{t'(\overline{\tau}') \in o} O^*(t'(\overline{\tau}')) \Big)$$

$$\text{with } \langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle \in LT$$

Note that $O^*(t(\overline{\tau}))$ is always finite due to the finiteness of the landmark table, even for cyclic method definitions.

Considering the previous example, the closures for the three abstract tasks of the planning problem Π are as follows: $O^*(t_1(c_1)) = O(t_1(c_1)) \cup O(t_3(c_2)) \cup O(t_3(c_1))$, $O^*(t_3(c_2)) = O(t_3(c_2))$, and $O^*(t_3(c_1)) = O(t_3(c_1))$.

Definition 10 (Landmark-aware strategy lm_1^*) *Given the prerequisites from Def. 8, the modification ordering function lm_1^* orders a plan modification $\mathtt{m}_i$ before $\mathtt{m}_j$ if and only if $\mathtt{m}_i$ addresses $\mathtt{f}_i$, $\mathtt{m}_j$ addresses $\mathtt{f}_j$, and*

$$\sum_{o \in O^*(t_i(\overline{\tau}_i))} |o|_{LT} < \sum_{o \in O^*(t_j(\overline{\tau}_j))} |o|_{LT}$$

So far, the "expansion effort" is defined in terms of decompositions that have to be applied until a solution is obtained. The following strategies take into account that primitive tasks contribute to the costs for developing the current plan into a solution, as well. The cost measure is thereby a uniform one: Solving the flaws affecting a primitive task is regarded as expensive as the expansion of an abstract one.

Definition 11 (Landmark-aware strategy lm_2) *Given the prerequisites from Def. 8, the modification ordering function lm_2 orders a plan modification $\mathtt{m}_i$ before $\mathtt{m}_j$ if and only if $\mathtt{m}_i$ addresses $\mathtt{f}_i$, $\mathtt{m}_j$ addresses $\mathtt{f}_j$, and*

$$\sum_{o \in O(t_i(\overline{\tau}_i))} |o| < \sum_{o \in O(t_j(\overline{\tau}_j))} |o|$$

Like we did for the landmark-aware strategy lm_1, we define a variant for strategy lm_2 that examines the transitive closure of the optional sets.

Definition 12 (Landmark-aware strategy lm_2^*) *Given the prerequisites from Def. 8, the modification ordering function lm_2^* orders a plan modification $\mathtt{m}_i$ before $\mathtt{m}_j$ if and only if $\mathtt{m}_i$ addresses $\mathtt{f}_i$, $\mathtt{m}_j$ addresses $\mathtt{f}_j$, and*

$$\sum_{o \in O^*(t_i(\overline{\tau}_i))} |o| < \sum_{o \in O^*(t_j(\overline{\tau}_j))} |o|$$

Since the landmark information can be extracted from any domain model and problem in an automated pre-processing step, the above strategies are conceptually domain- and problem-independent heuristics. In addition, they are independent from the actual plan generation procedure, hence their principles can be incorporated into any refinement-based hierarchical planning system.

5 Evaluation

We evaluated the performance of our novel strategies on four hierarchical planning domains along two dimensions: (1) we compared the time needed to find a solution in comparison to conventional hierarchical search strategies and (2) these benchmark tests were done on both the original planning domains and the corresponding ones that resulted from applying our landmark-based domain reduction technique (Elkawkagy, Schattenberg, and Biundo 2010). Hence, we base our evaluation on the same benchmark problems, but include two additional domains.

Conventional Hierarchical Search Strategies

For the strategies *SHOP* and *UMCP*, we used plan and modification ordering functions that induce the search strategies of these planning systems: In the UMCP system (Erol, Hendler, and Nau 1994), plans are primarily transformed into completely primitive plans in which causal interactions are dealt with afterwards. The SHOP strategy (Nau et al. 2003) prefers task expansion for the abstract tasks in the order in which they are to be executed.

In all other strategies the plan ordering function *Fewer Modifications First (fmf)* was used. It prefers plans for which a smaller number of refinement options is found, thereby implementing the least commitment principle on the plan ordering level. For the comparison to our landmark-aware modification ordering functions, we also conducted experiments with the following modification ordering functions:

The *Expand-Then-Make-Sound (ems)* procedure (McCluskey 2000) alternates task expansion with other modifications, which results in a "level-wise" concretization of plan steps. We also include the well-established *Least Committing First (lcf)* paradigm, a generalization of POCL strategies, which prefers those modifications that address flaws for which the smallest number of alternative solutions is available. HotSpot strategies examine plan components that are affected by multiple flaws, thereby quantifying to which extent solving one deficiency may interfere with the solution options for coupled components (Schattenberg, Bidot, and Biundo 2007). The *Direct Uniform HotSpot (du-HotSpot)* strategy avoids addressing flaws that refer to HotSpot plan components, and the *Direct Adaptive HotSpot (da)* strategy does so by increasing problem-specific weights on binary combinations of flaw types that occur in the plan. Finally, the *HotZone* strategy takes structural connections between HotSpots into account and tries to avoid modifications that deal with these clusters.

Benchmark Problem Set

We conducted our experiments on three well-established planning domains plus a domain taken from an ongoing research project. *Satellite* is a benchmark from the International Planning Competition (IPC) for non-hierarchical planning. The hierarchical encoding of this domain regards the original primitive operators as implementations of abstract observation tasks. The domain model consists of 3 abstract and 5 primitive tasks, and includes 8 methods. *WoodWorking*, also originally defined for the IPC in a non-hierarchical manner, specifies the processing of raw wood into smooth and varnished product parts. It uses 13 primitive tasks, 6 abstract tasks, and 14 methods. *UM-Translog* is a hierarchical planning domain that supports transportation and logistics. It shows 21 abstract and 48 primitive tasks as well as 51 methods. In addition to that, we also employed the so-called *SmartPhone* domain, a new hierarchical planning domain that is concerned with the operation of a smart phone by a human user, e.g., sending messages and creating contacts or appointments. SmartPhone is a rather large domain with a deep decomposition hierarchy, containing 50 abstract, 87 primitive tasks, and 94 methods.

Evaluation of Experimental Results

Tab. 1 shows the time required to solve the problems in our benchmark set for solving the original planning problem specification and the problem posed in a reduced domain model (Elkawkagy, Schattenberg, and Biundo 2010), respectively. By doing so, we evaluate the search guidance power of our landmark-aware strategies in relation to the domain reduction preprocessing technique.

In the UM-Translog domain (cf. Tab. 1a), at least one of our landmark-aware strategies belongs to the two best performing search strategies in all problem instances, at which lm_1 is clearly dominating the other landmark-based strategies; in fact, in 6 of 14 cases it has the best performance, and in 11 of 14 cases it is one of the two best strategies. This is quite surprising, because the landmark table does not reveal any information about causal dependencies on the primitive task level and the strategies hence cannot provide a focused guidance. A well-informed selection of the decomposition refinements obviously compensates for poor choices on the causality issues.

Our strategies show a similar behavior in the WoodWorking domain (cf. Tab. 1b): In all problems but one either lm_1 or lm_1^* is the best of the evaluated strategies. The similarity of the results to the ones in the UM-Translog domain is not surprising to us, as the depths of their decomposition hierarchies are similar.

The SmartPhone domain (cf. Tab. 1c) is the domain with the deepest decomposition hierarchy. It therefore carries the most information that is exploitable for landmarks, which results in well-informed landmark-aware strategies. Not surprisingly, in all but one problem instances one of our landmark-aware strategies performed best or second-best; in half of the instances, they performed best *and* second-best. The best result achieved lm_1, which was the best strategy in four of five instances and the second-best in another one.

On the Satellite domain (cf. Tab. 1d) our landmark-aware strategies do not clearly dominate any other strategy. Obviously, there is hardly any landmark information available due to the very shallow decomposition hierarchy in this domain. However, no other strategy in this domain dominated any landmark-aware strategy; thus, all evaluated strategies can be regarded as equally good.

An interesting facet of almost all problem instances (even among different domains) is that while the strategies lm_1^*/lm_2^* are the better informed heuristics they repeatedly perform worse than lm_1/lm_2. The same anomaly occurs when comparing lm_2/lm_2^* with the more abstract but also more successful lm_1/lm_1^*. We suppose these phenomena result from two sources: First, the random choice of ground candidates for the lifted task instances is relatively unreliable. This effect gets amplified by traversing along the landmark closures and into the primitive task level. Second, the most important choice points are on the early decomposition levels, i.e., once a method has been chosen for implementing an abstract task, this refinement puts more constraints on the remaining decisions than the strategy can infer from the feasibility analysis underlying the landmark table.

6 Conclusion

We introduced four novel search strategies for hierarchical planning which base their heuristic guidance on landmark information. We ran experiments on several benchmark domains and compared their performance with the performance of standard search heuristics from the literature. The results show that our landmark-aware strategies outperform the established ones on almost all problems with a deep decomposition hierarchy.

ACKNOWLEDGEMENTS

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 “Companion-Technology for Cognitive Technical Systems” funded by the German Research Foundation (DFG).

References

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief – a preliminary report on combining state abstraction and HTN planning. In *Proc. of ECP 2001*, 157–168.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *Proc. of ECAI 2010*, volume 215, 329–334. IOS Press.

Elkawkagy, M.; Schattenberg, B.; and Biundo, S. 2010. Landmarks in hierarchical planning. In *Proc. of ECAI 2010*, volume 215, 229–234. IOS Press.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. of AIPS 1994*, 249–254. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proc. of ICAPS 2009*, 162–169. AAAI Press.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *Proc. of ICAPS 2008*, 222–231. AAAI Press.

McCluskey, T. L. 2000. Object transition sequences: A new form of abstraction for HTN planners. In *Proc. of AIPS 2000*, 216–225. AAAI Press.

Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *JAIR* 20:379–404.

Porteous, J.; Sebastia, L.; and Hoffmann, J. 2001. On the extraction, ordering, and usage of landmarks in planning. In *Proc. of ECP 2001*, 37–48.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Schattenberg, B.; Bidot, J.; and Biundo, S. 2007. On the construction and evaluation of flexible plan-refinement strategies. In *Proc. of KI 2007*, 367–381. Springer.

Tsuneto, R.; Nau, D. S.; and Hendler, J. A. 1997. Plan-refinement strategies and search-space size. In *Proc. of ECP 1997*, volume 1348, 414–426. Springer.

Zhu, L., and Givan, R. 2004. Heuristic planning via roadmap deduction. In *IPC-4 Booklet*, 64–66.

Table 1: These results show the impact of the deployed modification ordering functions on the planning process. While SHOP and UMCP denote strategy function combinations that simulate the respective search procedures, all other strategy implementations use *fmf* as the plan ordering function. The columns labeled with **red** show the time in seconds needed to solve the problem if our domain reduction technique (Elkawkagy, Schattenberg, and Biundo 2010) is used, whereas columns labeled with **org** show the time needed to solve the *original* (unreduced) problem, respectively. All times include time for pre-processing. All values are the arithmetic means over three runs. Dashes indicate that no solution was found within a limit of 150 minutes. The best result for a given problem is emphasized bold, the second best bold and italic.

(a) UM-Translog: the problems differ in number and kind of locations and/or number of parcels to transport.

Mod. ordering	#1		#2		#3		#4		#5		#6		#7	
function f^{ModOrd}	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**
lcf	1878	225	**198**	173	3020	209	598	470	187	118	5047	1278	267	322
HotZone	***473***	***196***	255	***117***	498	224	549	527	***149***	121	–	–	**171**	***137***
lm_1	**243**	**180**	***221***	135	***447***	**184**	**512**	***434***	**121**	***111***	–	1172	***190***	**122**
lm_1^*	1772	212	593	**112**	**370**	***205***	1592	**420**	657	**109**	–	***1162***	1002	140
lm_2	3311	255	754	123	1670	248	1659	464	716	162	–	**1128**	925	151
lm_2^*	846	226	839	148	991	238	1712	487	583	340	***4921***	1318	1755	**122**
UMCP	952	244	278	114	994	229	***529***	474	187	122	**4893**	1263	215	127
ems	2056	1048	984	262	2199	1806	1889	976	696	295	–	–	876	235
da-HotSpot	2414	1958	350	257	–	2030	4695	2077	589	352	–	2560	578	352
du-HotSpot	1319	775	859	460	987	1090	1904	1304	692	224	–	–	391	258
SHOP	1735	353	283	241	1911	274	–	–	5874	4012	–	4005	911	190

(b) WoodWorking domain: the problems define variations of parts to be processed.

Mod. ordering	#1		#2		#3		#4		#5	
function f^{ModOrd}	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**
lcf	2067	350	–	–	–	–	–	–	–	–
HotZone	–	–	–	–	–	418	–	–	–	–
lm_1	***96***	***55***	2396	1815	**171**	159	732	**184**	**564**	**197**
lm_1^*	**82**	**50**	**669**	**245**	614	**98**	1561	1395	2109	1245
lm_2	881	433	–	1259	–	362	–	–	–	–
lm_2^*	1359	403	–	–	–	367	1935	1514	–	893
UMCP	228	133	3207	2936	***259***	125	***618***	***356***	892	218
ems	415	298	–	1275	–	2457	–	2256	–	512
da-HotSpot	113	85	***968***	***828***	355	***110***	**328**	357	***573***	***201***
du-HotSpot	–	–	–	–	–	–	–	–	–	–
SHOP	–	–	–	–	–	–	–	–	–	3578

(c) SmartPhone domain: assisting the user in managing different daily-life tasks.

Mod. ordering	#1		#2		#3	
function f^{ModOrd}	**org**	**red**	**org**	**red**	**org**	**red**
lcf	63	40	–	159	***8455***	6827
HotZone	65	***33***	490	212	–	–
lm_1	***50***	**30**	**134**	**53**	–	**465**
lm_1^*	65	50	392	173	–	–
lm_2	60	50	***181***	**53**	–	***680***
lm_2^*	98	76	1632	327	–	697
UMCP	80	**30**	256	***115***	–	–
ems	107	52	235	148	–	–
da-HotSpot	**45**	43	–	203	**1747**	1041
du-HotSpot	52	46	638	166	–	3421
SHOP	95	73	–	–	–	–

(d) Satellite domain: a column labeled with "x — y — z" stands for a problem instance with x observations, y satellites, and z different modes.

Mod. ordering	1 — 1 — 1		1 — 2 — 1		2 — 1 — 1		2 — 1 — 2		2 — 2 — 1		2 — 2 — 2	
function f^{ModOrd}	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**	**org**	**red**
lcf	95	93	154	77	1551	1338	–	4069	–	**701**	–	–
HotZone	76	***64***	142	62	–	4764	–	–	–	1338	–	1114
lm_1	89	80	209	208	***767***	***652***	**458**	***400***	**802**	***785***	***3307***	362
lm_1^*	86	85	***54***	***43***	1024	969	***2617***	2569	***960***	813	–	1228
lm_2	132	86	151	140	–	5804	2816	**251**	–	–	–	965
lm_2^*	102	80	191	99	–	–	2636	2553	–	–	–	**161**
UMCP	91	91	**51**	**41**	2035	1336	4150	1894	1215	1097	**2517**	1270
ems	74	60	62	53	2608	2856	–	–	1756	1579	4484	***175***
da-HotSpot	***69***	67	85	78	2136	1131	–	1131	6850	6841	–	–
du-HotSpot	107	**49**	270	150	–	–	–	–	–	–	–	–
SHOP	**66**	67	113	111	**270**	**264**	–	–	–	1780	–	–

The following pages show the publication:

P. Bercher, D. Höller, G. Behnke, and S. Biundo. "User-Centered Planning". In: *Companion Technology – A Paradigm Shift in Human-Technology Interaction*. Ed. by S. Biundo and A. Wendemuth. Cognitive Technologies. In print. Springer, 2017. Chap. 5, pp. 79–100. ISBN: 978-3-319-43664-7. DOI: 10.1007/978-3-319-43665-4_5

https://link.springer.com/chapter/10.1007/978-3-319-43665-4_5

The final publication is available at link.springer.com

User-Centered Planning

Pascal Bercher, Daniel Höller, Gregor Behnke, and Susanne Biundo

Abstract User-centered planning capabilities are core elements of *Companion*-Technology. They are used to implement the functional behavior of technical systems in a way that makes those systems *Companion*-able – able to serve users individually, to respect their actual requirements and needs, and to flexibly adapt to changes of the user's situation and environment. This book chapter presents various techniques we have developed and integrated to realize user-centered planning. They are based on a hybrid planning approach that combines key principles also humans rely on when making plans: stepwise refining complex tasks into executable courses of action and considering causal relationships between actions. Since the generated plans impose only a partial order on actions, they allow for a highly flexible execution order as well. Planning for *Companion*-Systems may serve different purposes, depending on the application for which the system is created. Sometimes, plans are just like control programs and executed automatically in order to elicit the desired system behavior; but sometimes they are made for humans. In the latter case, plans have to be adequately presented and the definite execution order of actions has to coincide with the user's requirements and expectations. Furthermore, the system should be able to smoothly cope with execution errors. To this end, the plan generation capabilities are complemented by mechanisms for plan presentation, execution monitoring, and plan repair.

1 Introduction

Companion-Systems are able to serve users individually, to respect their actual requirements and needs, to flexibly adapt to changes of the user's situation and environment, and to explain their own behavior (cf. Chap. 1, the survey on *Companion*-Technology [19], or the work of the collaborative research centre SFB/TRR 62 [21]).

Pascal Bercher · Daniel Höller · Gregor Behnke · Susanne Biundo
Ulm University, Institute for Artificial Intelligence, e-mail: forename.surname@uni-ulm.de

A core element when realizing such systems is user-centered planning. This chapter presents various techniques we have developed and integrated to realize user-centered planning [18]. They are based on a hybrid planning approach [20] that combines key principles also humans rely on when making plans by refining complex tasks stepwise into executable courses of action, assessing the various options for doing so, and considering causal relationships.

Planning for *Companion*-Systems may serve different purposes, depending on the application for which the system is created. Sometimes, plans are just like control programs and executed automatically in order to elicit the desired system behavior; but sometimes they are made for humans. In the latter case, plans have to be adequately presented to the user. Since the generated plans impose only a partial order on actions, they allow for a highly flexible execution order. A suitable total order must be selected for step-wise presentation: it should coincide with the user's requirements and expectations. We ensure this by a technique that allows finding *user-friendly* linearizations [33].

In particular when planning for humans, a plan execution component must monitor the current state of execution so that the system can detect failures, i.e., deviations from the expected execution outcome. In such a case, the hybrid plan repair mechanism finds a new plan that incorporates the execution error [17, 9].

Companion-Systems assist users to complete demanding tasks, so the user may not understand the steps that the system recommends to do. In particular after execution errors, question might arise due to the presentation of a new plan. To obtain transparency and to increase the user's trust in the system, it is essential that it is able to explain its behavior. Therefor, the purpose of any action within a plan may be automatically explained to the user in natural language [53].

These user-centered planning capabilities of plan generation, plan execution and linearization, plan repair, and plan explanation are essential capabilities to provide intelligent user-assistance in a variety of real-world applications [18]. As an example, we integrated all those techniques in a running system that assists a user in the task of setting up a complex home theater [34, 9, 15]. The respective system and, in particular, the integration of the user-centered planning capabilities with a knowledge base and components for user interaction is described in Chap. 24. Here, we focus on the underlying planning capabilities and explain them in detail. We use the planning domain of that application scenario as a running example.

The hybrid planning framework is explained in Sect. 2. Section 3 is devoted to plan execution. Plan execution consists of various key capabilities when planning for or with humans: the monitoring of the executed plans to trigger plan repair in case of arising execution errors (explained in Sect. 4), the linearization of plans to decide which plan step to execute next, and the actual execution of the next plan step, which includes the adequate presentation to the user. Section 5 introduces the plan explanation technique that allows to generate justifications for any plan step questioned by the user. Finally, Sect. 6 concludes the chapter.

2 Hybrid Planning Framework

Hybrid planning [36, 20] fuses Hierarchical Task Network (HTN) planning [27] with concepts known from Partial-Order Causal-Link (POCL) planning [41, 50].

The smooth integration of hierarchical problem solving (inherited from HTN planning) with causal reasoning (inherited from POCL planning) provides us with many capabilities that are beneficial when planning for or with humans:

HTN Planning. In HTN planning, problems are specified in terms of abstract activities one would like to have accomplished. To do so, they have to be refined step-wise into more specific courses of action that can be executed by the user. This provides us with certain benefits:

- First of all, a domain expert has more freedom in modeling a domain. Often, expert knowledge is structured in a hierarchical way. Hence, it is often known to the expert what actions need to be taken in which order to accomplish some high-level goal. Such knowledge can easily be modeled by introducing a hierarchy among the available actions. Many real-world application scenarios are hence modeled using hierarchical planning approaches such as hybrid planning or the SHOP approach [44, 39, 18]. Further, a domain modeler can be assisted in the task of creating a hierarchical domain model by techniques that automatically infer abstractions for hierarchical planning [7].
- That action hierarchy may then be exploited for generating and improving explanations [53]. When the user wants to know about the purpose of a presented action during execution, the hierarchy can be used to come up with a justification.
- The hierarchy defined on the actions may also be exploited to come up with plausible linearizations of plans [33]. The actions in the plans are presented to a user one-by-one. Some linearization might be more plausible to a user than others. So, presenting those actions close to each other that "belong to each other" with respect to the action hierarchy might achieve reasonable results.
- The way in which humans solve tasks is closely related to the way hierarchical problems are solved by a planning system. That makes it more natural to a user to be integrated into the planning process [5], as it resembles his or her idea of problem solving. The integration of the user into this decision making process is called *mixed initiative planning*. It is presented in Chap. 7.

POCL Planning. In POCL planning, problems are specified in terms of world properties that one would like to hold. The problem is solved via analyzing causal dependencies between actions to decide what action to take in order to fulfill a required goal. The way in which plans are found and how they are represented can be exploited in various ways:

- The causal dependencies between actions within a plan are explicitly represented using so-called causal links. Analogously and complementarily to the exploitation of the hierarchy, these causal relations can be analyzed and exploited to generate explanations about the purpose of any action within a plan [53].

- The causal structure of plans may be used to find plausible linearizations of the actions within a plan. For instance, given there are causal relationships between two actions, it seems more plausible to present them after each other before presenting another action that has no causal dependencies to either of them [33].
- Finally, the POCL planning approach also seems well-suited for a mixed initiative planning approach, since humans do not only plan in a hierarchical manner, but also via reasoning about which action to take in order to fulfill a requirement that has to hold later on.

Hybrid planning combines HTN planning with POCL planning; it hence features all before-mentioned user-centered planning capabilities.

2.1 Problem Formalization

A hybrid planning problem is given as a pair consisting of a domain model $\mathscr{D}$ and the problem instance $\mathscr{I}$. The domain describes the available actions required for planning. The problem instance specifies the actual problem to solve, i.e., the available world objects, the current initial state, the desired goal state properties, and an initial plan containing the abstract tasks that need to be refined.

More specifically, a domain is a triple $\mathscr{D} = \langle T_p, T_a, M \rangle$. T_p and T_a describe the *primitive* and *abstract tasks*, respectively. The primitive tasks are also referred to as *actions* – those can be executed directly by, and hence communicated to, the user. Actions are triples $\langle a(\bar{\tau}), pre(\bar{\tau}), eff(\bar{\tau}) \rangle$ consisting of a name a that is parametrized with variables $\bar{\tau}$, a parametrized precondition *pre* and the effects *eff*. As an example, Eq. (1) depicts the name and parameters of an action of the home assembly task (see Chap. 24) for plugging the audio end of a SCART cable into the audio port of an audio/video receiver. In the depicted action, the parameters are bound to constants, which represent the available objects – in the example domain those are the available hi-fi devices, cables, and their ports.

$$plugIn(\text{SCART-CABLE}, \text{AUDIO-PORT}, \text{AV-RECEIVER}, \text{AUDIO-PORT}) \qquad (1)$$

The precondition describes the circumstances under which the action can be executed, while the effects describe the changes that an execution has on the respective world state. Formally, preconditions and effects are conjunctions of literals that are defined over the variables $\bar{\tau}$. For instance, the (negative) literal $\neg used(\text{SCART-CABLE}, \text{AUDIO-PORT})$ is part of the precondition of the action depicted in Eq. (1). It describes that the audio port of the SCART cable may only be plugged into a port if it is currently not in use. The effects of that action mark the port as blocked. Abstract tasks syntactically look like primitive ones, but they are regarded to be not directly executable by the user. Instead, they are abstractions of one or more primitive tasks. That is, for any abstract task t, a so-called *(decomposition) method* $m = \langle t, P \rangle$ relates that task to a plan P that "implements" t [20, 13]. The set of all methods is given by M. The implementation (or legality) criteria ensure that t is a legal abstraction of the plan P, which can be verified by comparing the preconditions and effects of t with those of the tasks within P. One can also regard it the other way round: the implementation criteria ensure that only those plans may be used within a method that are actual implementations for the respective abstract task. That way, a human user (in that case the domain modeler) can be actively supported in the domain modeling process – independently of whether he or she uses a top-down or bottom-up modeling approach.

Plans are generalizations of action sequences in that they are only partially ordered. They are knowledge-rich structures, because causality is explicitly represented using so-called *causal links*. Formally, a plan P is a tuple $\langle PS, V, \prec, CL\rangle$ consisting of the following elements: the set PS is referred to as *plan steps*. Plan steps are uniquely labeled tasks. Thus, each plan step $ps \in PS$ is a tuple $l : t$, with l being a label symbol unique within P and t being a task taken from $T_p \cup T_a$. Unique labeling is required to differentiate identical tasks from each other that are all within the same plan. The set V contains the variable constraints that (non-)codesignate task parameters with each other or with constants. Codesignating a variable with a constant means to assign the respective constant to that variable. Codesignating two variables means that they have to be assigned to the same constant. Non-codesignating works analogously. The set $\prec$ is a strict partial order defined over $PS \times PS$. The causal links CL represent causal dependencies between tasks: each link $cl \in CL$ is a triple $\langle ps, \varphi, ps'\rangle$ representing that the literal φ is "produced" by (the task referenced by) ps and "consumed" by ps'. Due to that causal link, the precondition literal φ of ps' is called *protected*, since the solution criteria ensure that no other task is allowed to invalidate that precondition anymore (see Solution Criterion 2c given below).

A problem instance $\mathscr{I}$ is a tuple $\langle C, s_{init}, P_{init}, g\rangle$ consisting of the following elements: the set C contains all available constants. The conjunction s_{init} of ground positive literals describes the initial state. We assume the so-called *closed world assumption*. That is, exactly the literals in s_{init} are assumed to hold in the initial state, while all others are regarded false. The conjunction of (positive and negative) literals g describes the goal condition. All these goal state properties *must* hold after the execution of a solution plan. Hence, g implicitly represents a set of world states that satisfy g. In particular when planning for humans, not all goals are necessarily mandatory. Instead, some of them might only be *preferred* by the user. That is, while some goals might be declared as non-optional (those specified by g), a user might also want so specify so-called soft goals that he or she would like to see satisfied, but that are regarded optional. A planner would then try to achieve those goals to increase plan quality, but in case a soft-goal cannot be satisfied, the planning process does not fail altogether. Some work has been done in incorporating such soft-goals into hierarchical planning in general [39, 55] and in hybrid planning in particular [8]. For the sake of simplicity, we focus on the non-optional goals in this book chapter. Note that this is not a restriction, since any planning problem with soft goals can be translated into an equivalent problem without soft goals [23, 37]. The initial plan P_{init} complements the desired goal state properties by the tasks that the user would like to have achieved. This plan may contain primitive tasks, abstract tasks, or both. In addition, it contains two special actions a_{init} and a_{goal} that encode the initial state and goal description, respectively. The respective encoding is done as usual in POCL planning: a_{init} is always the very first action in every refinement of P_{init}, while a_{goal} is always the very last. The action a_{init} has no precondition and uses s_{init} as effect[1], while a_{goal} uses g as precondition and has no effect.

Solution Criteria. Informally, a solution is any plan that is executable in the initial state and satisfies the planning goals and tasks, i.e., after the execution of a solution plan P_{sol}, g holds, and P_{sol} is a refinement of P_{init} thereby ensuring that the abstract activities the user should accomplish (specified in P_{init}) have actually been achieved. More formally, a plan P_{sol} is a solution if and only if two criteria hold:

1. P_{sol} is a refinement of P_{init}. That is, one must be able to obtain P_{sol} from P_{init} by means of the application of the following refinement operators:

   a. ***Decomposition.*** Given a plan $P = \langle PS, V, \prec, CL \rangle$ with an abstract plan step $l : t \in PS$, the decomposition of the abstract task t using a decomposition method $m = \langle t, P' \rangle$ results in a new plan P'', in which $l : t$ is removed and replaced by P'. Ordering and variable constraints, as well as causal links pointing to or from $l : t$, are inherited by the tasks within P' [13]. This is a generalization of Def. 3 by Geier and Bercher [29] for standard HTN planning without causal links. This decomposition criterion ensures that the abstract tasks specified in P_{init} are accomplished by any solution. That criterion is the reason why HTN or hybrid planning is undecidable in the general case [27, 29, 1, 13]. It also makes the verification of plans (i.e., answering "is the given plan a valid solution to the given problem?") hard (NP-complete) even under severe restrictions [6, 13].

   b. ***Task Insertion.*** In hybrid planning, both primitive and abstract tasks may be inserted into a plan. Note that it is optional whether this feature is allowed or not. Allowing or disallowing task insertion might influence both the complexity of solving the planning problem [29, 2] and of the solutions themselves [31, 32]. Allowing task insertion allows for more flexibility for the domain modeler, as it allows to define partial hierarchical models [36, 2]. That is, the domain modeler does not need to specify decomposition methods that ensure that any decomposition is an executable solution, as the planner might insert tasks to ensure executability. Thus, allowing task insertion moves some of the planning complexity from the modeling process (which is done by a user/domain expert) to the planning process (which is done automatically).

   c. ***Causal Link Insertion*** **and** ***Ordering Insertion.*** Given two plan steps ps and ps', within a plan, a causal link can be inserted from any literal in ps's effect to any (identical) literal in the precondition of ps'. The parameters of the two literals become pairwise codesignated. Also an ordering constraint may be

[1] More technically, it uses not just s_{init} as effect, but – because s_{init} consists only of positive literals due to the closed world assumption – also all negative ground literals that unify with any negative precondition that are not contradicting s_{init}. Otherwise, there might be a negative task precondition literal that could not be protected by a causal link rooting in the initial state.

inserted between ps and ps'. Both these refinement options are inherited from standard POCL planning. They are a means to ensure the executability of plans [41, 50].

2. P_{sol} is executable in the initial state s_{init} and, after execution of that plan, the goal condition g is satisfied. Since s_{init} and g are encoded within P_{init} by means of the two special actions a_{init} and a_{goal}, respectively, and because P_{sol} is a refinement of P_{init} due to Solution Criterion 1, both planning goals can be achieved by using standard POCL solution criteria. Thus, $P_{sol} = \langle PS_{sol}, V_{sol}, \prec_{sol}, CL_{sol} \rangle$ is executable in s_{init} and satisfies g if and only if:

    a. ***All tasks are primitive and ground.*** Only primitive actions are regarded executable. Grounding is required to ensure unique preconditions and effects.
    b. ***There are no open preconditions.*** That is, for each precondition literal φ of any plan step $ps \in PS_{sol}$ there is a causal link $\langle ps', \varphi, ps \rangle \in CL_{sol}$ with $ps' \in PS_{sol}$ thereby protecting φ.
    c. ***There are no causal threats.*** We need to ensure that the literals used by the causal links are actually "protected". This is the case if there are no so-called *causal threats*. Within a primitive ground plan $P = \langle PS, V, \prec, CL \rangle$, a plan step ps is threatening a causal link $\langle ps', \varphi, ps'' \rangle \in CL$ if and only if the set of ordering constraints allows ps to be ordered between ps' and ps'' (that is, $\prec \cup \{(ps', ps), (ps, ps'')\}$ is a strict partial order) and ps has an effect $\neg\varphi$.

2.2 Finding a Solution

Hierarchical planning problems may be solved in many different ways [4], hence various hierarchical planning systems and techniques exist, such as SHOP/SHOP2 [43], UMCP [26], or HD-POP [52, edition 1, p. 374–375], to name just a few.

We follow the approach of the HD-POP technique. The resulting planning system, PANDA [14, Alg. 1], performs heuristic search in the space of plans via refining the initial plan P_{init} until a primitive executable plan has been obtained. The algorithm basically mimics the allowed refinement options: it decomposes abstract tasks (thereby introducing new ones into the successor plan), inserts new tasks from the domain (if allowed; cf. Solution Criterion 1b), and inserts ordering constraints and causal links to ensure executability. Hierarchical planning is quite difficult. In the general case, it is undecidable, but even for some quite restricted special cases, it is still at least PSPACE-hard [27, 29, 1, 2, 13]. During search, that hardness corresponds to the choice of which task to insert and which decomposition method to pick when decomposing an abstract task. In the approach taken by PANDA, these questions are answered by heuristics: each candidate plan is estimated in terms of the number of required modifications to refine it into a solution, or by means of the number of actions that need to be inserted for the same purpose [14].

For standard POCL planning, i.e., in case the initial plan P_{init} does not contain abstract tasks, there are basically two different kinds of heuristics. The first kind bases on delete-relaxation[2], as this reduces the complexity of deciding the plan existence problem from PSPACE to P or NP, depending on the presence of negative preconditions [22] and whether the actions in the domain and the given plan become delete-relaxed or just those in the domain [11]. The respective heuristics are the *Add Heuristic for POCL planning* [57], the *Relax Heuristic* [46], and a variant of the latter based on partial delete-relaxation, called *SampleFF* [11]. The second kind of heuristics is not just one single POCL heuristic, but a technique that allows to *directly* use heuristics known from state-based planning in the POCL planning setting [10]. The technique encodes a plan into a classical (i.e., non-hierarchical) planning problem, where the POCL plan is encoded within the domain.

The idea of delete-relaxation has also been transferred to hierarchical planning. Here, the complexity of the plan existence problem is reduced from undecidable to NP or P, depending on various relaxations [3]. There is not yet an implementation of that idea, however. Instead, we developed the so-called *task decomposition graph* that is a relaxed representation of how the abstract tasks may be decomposed [25, 24]. That graph may both be used for pruning infeasible plans from the search space (i.e., plans that cannot be refined into a solution) [25] and for designing well-informed heuristics for hierarchical and hybrid planning [24, 14].

3 Plan Execution

In most real-world application domains, the effect of actions is not fully deterministic, though there is often an outcome that can be regarded as the intended or standard effect. Since *Companion*-Systems flexibly adapt to any changes in the user's situation and environment, they must be able to detect and deal with unforeseen effects. The sub system that monitors the environment and detects state changes that conflict with the current plan is called *execution monitor* and described in Sect. 3.1. When a state deviation is detected that may cause the current plan to fail, the *plan repair* component is started. The plan repair mechanism is introduced later on in Sect. 4. Solution plans are not totally ordered: they include only ordering constraints that are necessary to guarantee executability. Thus it is likely that there is more than one linearization of the solution. The *plan linearization* component is responsible to decide which one is most appropriate to be presented to a user. This functionality is described in Sect. 3.2. What it means to execute a single plan step, and how it may be done, is described in Sect. 3.3.

[2] Delete-relaxation means to ignore negative literals in the effects and, optionally, in the preconditions of any action.

3.1 Monitoring

As given above, the monitoring compares changes that have been detected in the environment with the intended effect of a started action. When differences are detected, it must not necessarily be a problem for the execution of the current plan, so the monitoring has to decide whether repair (see Sect. 4) is initiated or not. The decision may be based on the set of *active* causal links. A causal link is active if and only if its producer has been executed while the consumer has not. When there is an active link on a literal that has changed, repair is started (see Fig. 1).

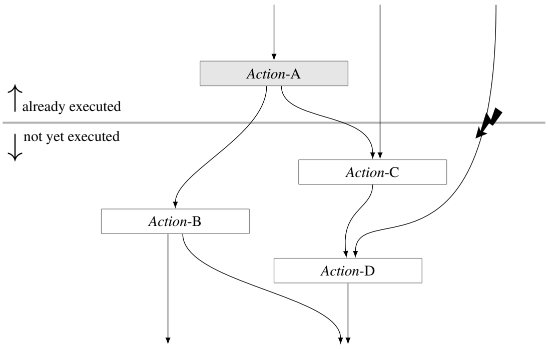


Fig. 1 The figure shows how an unexpected state deviation influences the execution of the remaining actions. The horizontal line indicates the execution horizon. An execution error flipped the truth value of the literal protected by the right-most causal link. Because that causal link crosses the execution horizon, the causal link's consumer (Action-D) might not be executable anymore.

Intuitively, this means that (a part of) the precondition of the consumer should have been fulfilled by the producer, but this has not been successful. Now there is no guarantee that the precondition of the consumer is fulfilled (i.e., a causal link that supports it) and plan execution may fail. There are special cases, however, where the currently executed sequence of actions is still executable although there is a causal link that is violated (a valid POCL plan could hence be found by simply choosing a new producer for the invalidated causal link within that plan). Although in that case the user could proceed executing that action sequence, plan repair must be initiated, since the respective causal link may be mandatory: in case it has not been inserted by the planner as a means to ensure executability, but if it comes from the domain (specified within a plan referenced by a decomposition method) or from the initial plan, it may not be changed. Such links may be intended by the modeler to protect certain properties during execution (referred to as prevail conditions) and are thus not allowed to be removed.

So, when ever a condition of an active causal link is violated, the plan monitoring initiates plan repair. It creates an altered plan (if there is one) that is able to deal with unexpected changes and fulfills the constraints given in the model.

The approach given above is able to deal with unforeseen changes of the environment and minimizes the computational effort that is necessary. Plan repair is only started if active causal links are violated. However, there are situations where it would be beneficial to start the repair mechanism even in cases where no active causal links are violated and, hence, the plan is still executable. Consider, e.g., the case where the unforeseen change does not result in any violated causal link, but at the same time causes the original goal condition to become true. In that setting, the planning problem would be solved when no further actions are executed[3]. Without starting repair, the user had to proceed executing the plan. Though this is a good reason to start plan repair as often as there is enough time to wait for the new plan, there are also reasons to continue the execution of the original plan: in case there is no notable problem with the plan currently executed it might confuse the user when its execution is canceled to proceed another plan. The question when to repair could be answered by an empirical evaluation.

3.2 Plan Linearization

As given in the introduction of this section, plans generated by the planning system are only partially ordered. They include only the ordering constraints that are included in the model and those that have to be included to guarantee that a goal state is reached after execution. This makes the execution most flexible, since it commits only on necessary constraints. In many situations, it is necessary to choose a linearization of that partial order for plan execution. When plans are executed by a machine, like a smartphone or robot, it may not matter which of its linearizations is executed. However, whenever humans are involved in plan execution, the low commitment of the ordering given in the plan can be exploited to choose the linearization that is most suitable for the specific user in the current situation. Consider a user who has to achieve two tasks that are not related in any sense. This scenario is likely to result in a plan with two lines of action that are not interrelated. It is no problem to execute the first step of the first line, then the first step of the second line, and so on. However, it might be much more intuitive for the user to finish the first line before starting the second one (or vice versa). The overall process, committing on some ordering constraints during planning and determining the other ordering relations during post processing, can be seen as a model that consists of two parts.

There are several objectives for the linearization that may be competing. This could be the convenience of the user during execution, to optimize a metric that can be measured (e.g., execution time) or to imitate human behavior. Since *Companion*-Systems need to adapt to the specific user and its current situation, finding a user-friendly and maybe user- and situation-specific linearization is another point where adaptivity may come to light. As a starting point for situation- and user-specific strategies, we identified three domain-independent strategies to linearize plans [33].

All of them exploit knowledge that is included in the plan or the domain definition to linearize plans:

[3] Assuming there are no not yet executed actions that are inserted due to the underlying action hierarchy, cf. Solution Criterion 1a.

1. **Parameter Similarity.** In the home theater domain (see Chap. 24) it seems reasonable to complete all actions involving a specific device before starting on another. It is a feasible design decision of the modeler to pass on the devices as parameters to an action (though there are other ways to model the domain), as it is the case for the example action in Eq. (1). A parameter-based strategy would exploit this: it orders plan steps in a way that maximizes successive actions that share constants in their parameter set [33, Section 4.1].
2. **Causal Link Structure.** The causal link structure of a plan represents which effect of a plan step fulfills a certain precondition of another. The planning procedure is problem driven, i.e., there is no needless causal link in the plan. Therefore this is also a valuable source of linearization information, because the user may keep track of the causality behind steps that are executed. A strategy based on this structure orders the steps in a way that minimizes the distance between producer and consumer of a causal link. Besides the decisions of the domain modeler, this strategy also depends on the planning process [33, Section 4.2].
3. **Decomposition Structure.** Since the planning domain is commonly modeled by a human domain designer, it is reasonable to assume that tasks that are introduced by a single method are also semantically related. Generalizing this assumption, tasks that have a short distance in the tree of decompositions that spans from the initial task network to the actual plan steps are supposed to be semantically closer related than tasks that have a long distance. This property can be used for plan linearization. In this form, it depends on both the domain and the planning process. When using the task decomposition graph instead, it only depends on domain properties [33, Section 4.3].

As given above, all strategies depend on the planning domain, the planning system, or both. Thus it is possible to model the same application domain in such a way that they work well or poorly. Consider, e.g., the strategy based on parameter similarity in a propositional domain – there is no information included that could be used for linearization.

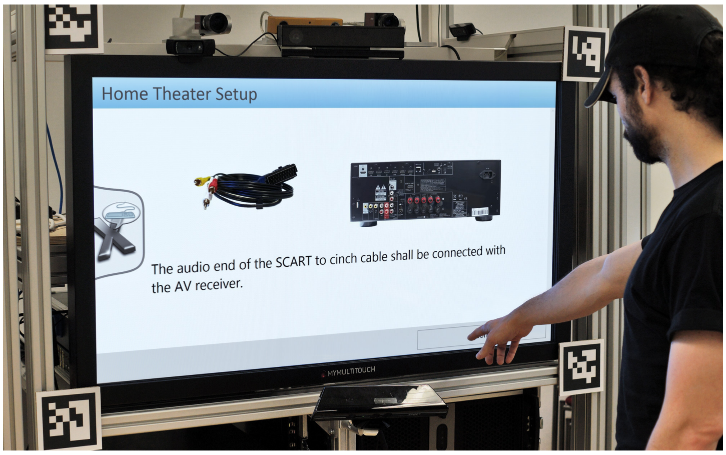
The given strategies can be used to pick the next plan step from a set of possible next actions (those where all predecessors in the ordering relation have already been finished), i.e., for a local optimization. Another possibility is to optimize them globally over the linearization of the whole plan. They can also be used as starting point for a domain-specific strategy.

3.3 Plan Step Execution

There are several possibilities on how to proceed when a single plan step has been selected for execution. In some cases, the action is just present due to technical reasons and nothing has to be done for its execution. Consider, e.g., the actions a_{init} and a_{goal}. Their purpose is to cause a certain change during the planning process and it is likely that they can be ignored by the execution system, although reaching action a_{goal} could trigger a notification that states the successful plan completion.

A second possibility is that actions control some part of the system. These are executed internally, but are not necessarily required to be communicated to the user. They may cause, for example, a light to be switched on/off, or a door to open/close, or adding a new entry to be added in a calendar.

Besides these possibilities, there are actions that have to be communicated to the user, as he or she is the one that carries them out or because the presentation itself is the desired purpose. Such actions can be easily communicated to the user by relying on additional system components taken from dialog management (Chap. 9) and user interaction (Chap. 10), as explained in Chap. 24. For that purpose, each action has an associated dialog model that specifies how it may be presented to a user [47, 16]. The dialog model may itself be structured in a hierarchical manner to enable the presentation of an action with a level of detail that is specific to the individual user. So, depending on the user's background knowledge, the action may be presented with more or less details [48]. The resulting information is sent to the fission component [35] that is responsible for selecting the adequate output modality (Chap. 10). For this, each action may have a standard text template associated with it, which can be used for visualization. Further, each constant used by an action parameter can be associated with respective graphics or videos. In Fig. 2 we see how the action given in Eq. (1) may be presented to a user.



The audio end of the SCART to cinch cable shall be connected with the AV receiver.

Fig. 2: Here, we see how a single planning action can be presented to a human user.

4 Repairing Failed Plans

Companion-Systems have to adapt to changes in the current situation [19]. This is especially necessary when the execution of a plan fails. If the plan monitoring component (see Sect. 3.1) decides that – due to an execution failure – a new plan has to be found, there are two possibilities how this can be done:

- **Re-planning.** The plan at hand is discarded and the planning process is done from scratch. The changed environment is used as initial state and a new plan is found that transfers it into a state that fulfills the goal criteria.
- **Plan Repair.** The original plan is re-used and adapted to the needs of the changed situation. Thereby the unexpected changes of the environment have to be considered and to be integrated into the new plan.

Both approaches have several advantages and disadvantages. *Re-planning* enables the use of sophisticated planning heuristics. For some cases in classical planning, Nebel and Koehler showed that plan repair might be computationally more expensive than planning from scratch [45]. The system could come up with a completely new solution that has nothing in common with the original one, albeit a minor change would have resulted in a valid solution. Presenting a very different solution to a human user might cause confusion and reduce the user's trust in the system.

When a plan is *repaired*, the new plan might be more similar to the original solution. However, this strategy might increase computational complexity [45], prevents the planning system to find shorter/more cost-effective solutions; and an altered algorithm with effective heuristics that are able to deal with the altered planning problem has to be realized.

In HTN planning there is another aspect to consider: While in classical planning the already executed prefix of the original solution, followed by a completely new plan that reaches a goal state is a proper solution to the original problem, the combination may not be in the decomposition hierarchy of an HTN problem and thus violate Solution Criterion 1. There are circumstances that can be encoded into the decomposition hierarchy of an HTN planning problem that can not be ensured by preconditions and effects (see the expressivity analysis by Höller et al. [31, 32]). So it has to be assured that a repaired plan also fulfills the constraints that are introduced by the hierarchy.

We now introduce our approach for *re-planning* [9]. Although, from a theoretical point of view, it is classified as re-planning (because we do not try to repair the plan already found), it still combines aspects of both re-planning and plan repair. Aspects of repair are required to ensure that the plan prefix already executed is also part of any new solution that can cope with the execution error.

When the execution of a plan fails, a plan repair problem is created. Its domain definition is identical to that of the original problem, while the problem instance is adapted. It includes an additional set of obligations O, i.e., $\mathscr{I} = \langle C, s_{init}, P_{init}, O, g \rangle$. Obligations define which commitments that were made in the original plan have to be present in the new one. To ensure that they are fulfilled, we extend the solution criteria in such a way that all obligations need to be satisfied. There are obligations of the following kind:

- **Task Obligations.** These obligations ensure that a certain plan step (i.e., action) is present in the new solution. A task obligation is included into the problem for every step of the original plan that has already been executed. To overcome the unexpected environment change, a special task obligation is added to the repair problem. It makes sure that a new action is added that realizes the unforeseen changes of the environment. Therefore it has the detected change as its effect. This action is called *process* [17] and is introduced after the executed prefix of the original plan.

- **Ordering Obligations.** Ordering obligations define ordering constraints between the obligated task steps.

Obligations from the given classes are combined in a way ensuring that the executed prefix of the original plan is also a prefix of any new plan. The process is placed exactly behind this prefix to realize the detected change of the world. So the new plan can cope with the unforeseen changes of the environment.

The additional constraints (i.e., the obligations) require some small alterations of the planning procedure. Given an unsatisfied obligation, the algorithm needs to provide possible refinements therefor: unsatisfied task obligations can be addressed via task insertion or decomposition and marking a task within a plan as one of those already executed. Ordering obligations are straight-forward.

We have also developed a *repair* approach for hybrid planning [17, 18]. It starts with the original planning problem and the set of refinements applied to find the original solution. As it is the case for our *re-planning* approach, the obligations are part of the planning problem as well to ensure that the execution error is reflected and the actions already executed are part of the repaired solution. In contrast to standard repair, the algorithm tries to re-apply all previously applied refinements. It only chooses different refinements where the particular choice leads to a part of the plan that cannot be executed anymore due to the execution failure.

5 Plan Explanation

Plans generated via automated planning are usually fairly complex and can contain a large amount of plan steps and causal links between them. If the decisions of a *Companion*-System are based upon such plans, its user may not immediately understand the behavior of the system completely. In the worst case, he or she might even reject the system's suggestions outright and stop using it altogether. In general, unexpected or non-understandable behavior of a cognitive system may have a negative impact on the trust in human-computer relationship [42], which in turn is known to have adverse effects on the interaction with the user [49]. To avert this problem, a system should be able to explain its decisions and internal behavior [40, 12]. If a planner is the central cognitive component of the system, it has to be able to explain its decisions (i.e., the plan it has produced) to the user.

A first step towards user-friendly interaction and eliminating questions of the users even before they come up is an intelligent plan linearization component (see Sect. 3.2), which presents the whole plan in an easy to grasp step-by-step fashion. Obviously, this capability is not sufficient for complete transparency. Although the order in which actions are presented is chosen in such a way that it is intuitive for the user, he or she might still wonder about it or propose a rearrangement. The user might also be confused about the actual purpose of a presented action and ask why it is part of the solution in the first place. The hybrid plan explanation [53] technique is designed to convey such information to the user.

5.1 Generating Formal Plan Explanations

Usually, plan explanations are generated upon user request. Currently, the hybrid plan explanation technique supports two types of requests. The first inquires for the necessity of a plan step, i.e., “Why is action A in the plan?” or “Why should I do A?”. The second requests information on an ordering of plan steps, i.e., “Why must action A be executed before B?” or “Why can’t I do B after A?”. In both cases the explanation is based upon a proof in an axiomatic system Σ, which encodes the plan, the way it was created, and general rules how facts about the plan can be justified [53]. The request of the user is transformed into a fact F and an automated reasoner is applied to compute a proof for $\Sigma \vdash F$. This proof is regarded as the actual formal explanation of the fact the user has inquired and is – subsequently – transformed into natural language by a dialogue management component and presented to the user [53, 9]. Obtaining such a proof in a *general* first order axiomatic system is undecidable. In our case it is decidable, since all necessary axioms are horn-formulas, i.e., disjunctions of literals with at most one being positive. This allows for the application of the well-known SLD-resolution [38] to find proofs.

We now describe which axioms are contained in Σ and how the inference, i.e., obtaining the formal proof, can be done. The plan itself is encoded in Σ by several axioms, using two ternary predicates *cr* and *dr*, which describe the causal and hierarchical relations, respectively. For every causal link $\langle ps, \varphi, ps' \rangle$ in the plan, the axiom $cr(ps, \varphi, ps')$ is added to Σ. As described in Sect. 2, the plan to be explained has been obtained by applying a sequence of modifications, i.e., by adding causal links, ordering constraints, tasks, or by decomposing abstract tasks. Each used method m was applied to decompose some abstract plan step ps'. It adds a set of new plan steps PS (and ordering constraints and causal links) to the plan. For every such plan step $ps \in PS$ the axiom $dr(ps, m, ps')$ is added to Σ.

Explaining the Necessity of Plan Steps. To answer the first kind of question, axioms proving the necessity of a plan step must be defined. That necessity is described using the unary predicate n. Note that by “necessity” we do not refer to an absolute or global necessity of a plan step. We do not answer the question whether the respective action has to be part of any solution (such actions are called *action landmarks* [51, 58]). Answering this question is in general as hard as planning itself. Instead, we explain the *purpose* of the action: we give a chain of arguments explaining for which purpose that action is used within the presented plan.

All plan steps of the initial plan P_{init} (which includes the action a_{goal} that encodes the goal condition) are necessary by definition, since P_{init} describes the problem itself. Thus, $n(ps)$ is included as an axiom for every plan step ps of P_{init}. If a plan step ps is contained in the plan in order to provide a causal link for another necessary plan step, ps is also regarded necessary, as it establishes a precondition of a required action. A simple example application of this rule is the necessity of any action establishing one (or more) of the goal conditions. The information that a plan step establishes the precondition of another plan step is explicitly given in hybrid plans by causal links. Using the given encoding of causal links in Σ, we can formulate an axiom to infer necessity as follows:

$$\forall ps, \varphi, ps' : cr(ps, \varphi, ps') \land n(ps') \rightarrow n(ps) \tag{2}$$

A similar argument can be applied if a plan step ps has been obtained via decomposition. If a necessary abstract plan step ps' is decomposed into ps, then ps serves the purpose of refining ps'. Converted into an axiom this reads:

$$\forall ps, m, ps' : dr(ps, m, ps') \land n(ps') \rightarrow n(ps) \tag{3}$$

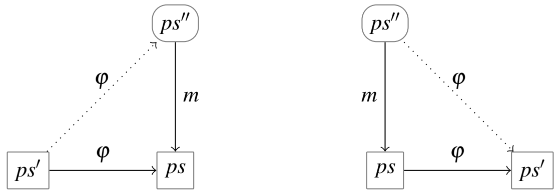
One can use both Axiom (2) and (3) to show the purpose of any plan step: it is either used to ensure the executability of another plan step (in this case, the first rule may be applied), or it is part of the plan because of decomposition (then, the second rule applies). Any chain of arguments (i.e., rule applications) will subsequently root in a plan step of the initial plan, i.e., the number of proof steps is always finite.

So far, the explanations based on causal dependencies (cf. Axiom (2)) do only rely on *primitive* plan steps. However, even these causality-based explanations could be improved when taking into account abstract tasks. E.g., the presence of the plan step *plugIn*(SCART-CABLE, AUDIO-PORT, AV-RECEIVER, AUDIO-PORT) should be explained as follows: it is necessary, as it is part of the abstract task *connect*(BLUERAY-PLAYER, AV-RECEIVER), which provides *signalAt*(AUDIO, AV-RECEIVER), which in turn is needed by the action *connect*(AV-RECEIVER, *TV*) to achieve the goal *signalAt*(AUDIO, *TV*). To obtain such explanations, *cr* predicates (i.e., causal links) involving abstract tasks must be inferred. Here, the idea is that if a plan step ps has an effect (or precondition) linked to some other plan step ps' that has been introduced into the plan by decomposing ps'', then ps'' is also linked to ps' as one of its primitive tasks generated the condition necessary for ps'. The axiomatic system Σ contains two further axioms, inferring these *cr* relations. Figure 3 contains a visual representation of both axioms.

$$\forall ps, m, ps'', ps' : dr(ps, m, ps'') \land cr(ps', \varphi, ps) \rightarrow cr(ps', \varphi, ps'') \tag{4}$$

$$\forall ps, m, ps'', ps' : dr(ps, m, ps'') \land cr(ps, \varphi, ps') \rightarrow cr(ps'', \varphi, ps') \tag{5}$$

Explaining the Order of Plan Steps. The second question a user might pose, i.e., why a plan step ps is arranged before some other plan step ps', has two possible answers. Either the order is contained in the plan presented to the user or it was chosen as part of the plan linearization process. In the latter case the system's answer could state that the order was chosen to obtain a plausible linearization and can be changed if the user wishes to. In the former case, again a proof for a fact is generated



(a) Visualization of Axiom (4) (b) Visualization of Axiom (5)

Fig. 3: The rectangular boxes depict primitive plan steps, the ones with rounded corners depict abstract plan steps. The arrows labeled with m indicate a performed decomposition using the method m. The arrows labeled with the literal φ indicate causal links, whereas the dotted ones are inferred by one of the Axioms (4) or (5).

and conveyed to the user. Necessary order between plan steps is encoded using the binary relation $<$. If the user poses the said question, the fact $ps < ps'$ is to be proven in Σ and its proof constitutes the formal explanation for the order's necessity. A necessary order can be caused by several reasons, each of which is described by an axiom. For the sake of brevity, we will only provide an intuition on these axioms, while the interested reader is referred to the work of Seegebarth et al. [53] for further details.

Orderings can be contained in the plans referenced by decomposition methods, thus they are necessary if the respective abstract task is. Further, an ordering constraint may be added to a plan if a causal threat is to be dissolved. Here, the necessity is based on the threatening plan step of the threat (cf. Solution Criterion 2c). Order is also implicitly implied by every causal link in the plan, and its necessity is based on the necessity of the consuming plan step of the link.

5.2 Verbalizing Plan Explanations

After having obtained a formal plan explanation, expressed by a proof in first-order logic, it has to be conveyed to the user in a suitable way. As a default-approach the explanation is transformed into text, which can be read to the user or displayed on a screen (see Fig. 2). To generate a natural language text, we use a pattern-based approach, an approach commonly used by automated theorem provers to present their proofs to humans [54, 30, 28]. Additionally one could use techniques similar to the Interactive Derivation Viewer [56], which uses both verbal and visual explanations.

Consider the example mentioned earlier in this section. The formal explanation in this case consists of one application of Axiom (3), two applications of the Axiom (2), one of Axiom (4), and two of Axiom (5). Resulting from this proof the following natural language text is generated:

> Plug the audio end of the SCART-to-Cinch cable into the AV Receiver to connect the Bluray Player with the AV Receiver. This provides that the AV Receiver has an audio signal,

needed to connect the AV Receiver with the TV. This provides that the TV has an audio signal, needed to achieve the goal.

We chose not to verbalize Axioms (4) and (5), as their application should be intuitively clear to the user. The remaining applications of Axiom (2) and (3) form a linear list. Each occurrence of Axiom (2) is translated into the text "This provides that $\langle\varphi\rangle$, needed to $\langle ps'\rangle$", where $\langle x\rangle$ denotes a domain-dependent verbalization of x. Likewise, each instance of Axiom (3) is translated into "Do this to $\langle ps'\rangle$" For the very first axiom in the explanation the begin of the sentences "This" and "Do this" are replaced with the verbalization of the action to be explained.

6 Conclusion

Flexible system behavior is essential when realizing *Companion*-Systems [19]. We summarized how different system capabilities supporting this design goal can be implemented using the hybrid planning approach, starting by the generation process that might integrate the user, the execution and communication of generated solutions, as well as discussing how to cope with unforeseen situations.

Though the current abilities of user-centered planning contributes valuable capabilities to the overall system, there are several promising lines of research for further improvements. Especially the problem of how plans are linearized [33] may offer further benefits for a convenient system. Another direction is a deeper explanation of system behavior [53, 9]. Here questions like "Why can't I use this action/method?" or an explanation on why a problem at hand has no solution may help the user. The overall explanation quality might also be improved by further integrating ontology- as well as plan-based explanations [7]. Another important matter in real-world applications is the presentation of different alternatives to reach a goal [12].

Acknowledgements This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 "*Companion*-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

1. Alford, R., Bercher, P., Aha, D.: Tight bounds for HTN planning. In: Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 7–15. AAAI Press (2015)
2. Alford, R., Bercher, P., Aha, D.: Tight bounds for HTN planning with task insertion. In: Proc. of the 25th Int. Joint Conf. on AI (IJCAI), pp. 1502–1508. AAAI Press (2015)
3. Alford, R., Shivashankar, V., Kuter, U., Nau, D.: On the feasibility of planning graph style heuristics for htn planning. In: Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 2–10. AAAI Press (2014)
4. Alford, R., Shivashankar, V., Kuter, U., Nau, D.S.: HTN problem spaces: Structure, algorithms, termination. In: Proc. of the 5th Annual Symposium on Combinatorial Search (SoCS), pp. 2–9. AAAI Press (2012)

5. Behnke, G., Höller, D., Bercher, P., Biundo, S.: Change the plan - how hard can that be? In: Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 38–46. AAAI Press (2016)
6. Behnke, G., Höller, D., Biundo, S.: On the complexity of HTN plan verification and its implications for plan recognition. In: Proc. of the 25th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 25–33. AAAI Press (2015)
7. Behnke, G., Ponomaryov, D., Schiller, M., Bercher, P., Nothdurft, F., Glimm, B., Biundo, S.: Coherence across components in cognitive systems – one ontology to rule them all. In: Proc. of the 25th Int. Joint Conf. on AI (IJCAI), pp. 1442–1449. AAAI Press (2015)
8. Bercher, P., Biundo, S.: A heuristic for hybrid planning with preferences. In: Proc. of the 25th Int. Florida AI Research Society Conf. (FLAIRS), pp. 120–123. AAAI Press (2012)
9. Bercher, P., Biundo, S., Geier, T., Hörnle, T., Nothdurft, F., Richter, F., Schattenberg, B.: Plan, repair, execute, explain - how planning helps to assemble your home theater. In: Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 386–394. AAAI Press (2014)
10. Bercher, P., Geier, T., Biundo, S.: Using state-based planning heuristics for partial-order causal-link planning. In: Advances in AI, Proc. of the 36th German Conf. on AI (KI), pp. 1–12. Springer (2013)
11. Bercher, P., Geier, T., Richter, F., Biundo, S.: On delete relaxation in partial-order causal-link planning. In: Proc. of the 25th Int. Conf. on Tools with AI (ICTAI), pp. 674–681. IEEE Computer Society (2013)
12. Bercher, P., Höller, D.: Interview with David E. Smith. Künstliche Intelligenz (2016). DOI 10.1007/s13218-015-0403-y
13. Bercher, P., Hller, D., Behnke, G., Biundo, S.: More than a name? on implications of preconditions and effects of compound htn planning tasks. In: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016), pp. 225–233. IOS Press (2016)
14. Bercher, P., Keen, S., Biundo, S.: Hybrid planning heuristics based on task decomposition graphs. In: Proc. of the 7th Annual Symposium on Combinatorial Search (SoCS), pp. 35–43. AAAI Press (2014)
15. Bercher, P., Richter, F., Hörnle, T., Geier, T., Höller, D., Behnke, G., Nothdurft, F., Honold, F., Minker, W., Weber, M., Biundo, S.: A planning-based assistance system for setting up a home theater. In: Proc. of the 29th Nat. Conf. on AI (AAAI), pp. 4264–4265. AAAI Press (2015)
16. Bertrand, G., Nothdurft, F., Honold, F., Schüssel, F.: CALIGRAPHI-creation of adaptive dialogues using a graphical interface. In: 35th Annual Computer Software and Applications Conf. (COMPSAC), pp. 393–400. IEEE (2011)
17. Bidot, J., Schattenberg, B., Biundo, S.: Plan repair in hybrid planning. In: Advances in AI, Proc. of the 31st German Conf. on AI (KI), pp. 169–176. Springer (2008)
18. Biundo, S., Bercher, P., Geier, T., Müller, F., Schattenberg, B.: Advanced user assistance based on AI planning. Cognitive Systems Research **12**(3-4), 219–236 (2011). Special Issue on Complex Cognition
19. Biundo, S., Höller, D., Schattenberg, B., Bercher, P.: Companion-technology: An overview. Künstliche Intelligenz (2016). DOI 10.1007/s13218-015-0419-3
20. Biundo, S., Schattenberg, B.: From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In: Proc. of the 6th European Conf. on Planning (ECP), pp. 157–168. AAAI Press (2001)
21. Biundo, S., Wendemuth, A.: *Companion*-technology for cognitive technical systems. Künstliche Intelligenz (2016). DOI 10.1007/s13218-015-0414-8
22. Bylander, T.: The computational complexity of propositional STRIPS planning. AI **94**(1-2), 165–204 (1994)
23. Edelkamp, S.: On the compilation of plan constraints and preferences. In: Proc. of the 16th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 374–377. AAAI Press (2006)
24. Elkawkagy, M., Bercher, P., Schattenberg, B., Biundo, S.: Improving hierarchical planning performance by the use of landmarks. In: Proc. of the 26th Nat. Conf. on AI (AAAI), pp. 1763–1769. AAAI Press (2012)

25. Elkawkagy, M., Schattenberg, B., Biundo, S.: Landmarks in hierarchical planning. In: Proc. of the 20th European Conf. on AI (ECAI), pp. 229–234. IOS Press (2010)
26. Erol, K., Hendler, J.A., Nau, D.S.: UMCP: A sound and complete procedure for hierarchical task-network planning. In: Proc. of the 2nd Int. Conf. on AI Planning Systems (AIPS), pp. 249–254. AAAI Press (1994)
27. Erol, K., Hendler, J.A., Nau, D.S.: Complexity results for HTN planning. Annals of Mathematics and AI **18**(1), 69–93 (1996)
28. Fiedler, A.: P.rex: An interactive proof explainer. In: Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR), pp. 416–420. Springer (2001)
29. Geier, T., Bercher, P.: On the decidability of HTN planning with task insertion. In: Proc. of the 22nd Int. Joint Conf. on AI (IJCAI), pp. 1955–1961. AAAI Press (2011)
30. Holland-Minkley, A.M., Barzilay, R., Constable, R.L.: Verbalization of high-level formal proofs. In: Proc. of the 16th Nat. Conf. on AI and the 11th Innovative Applications of AI Conf. (AAAI/IAAI), pp. 277–284. AAAI Press (1999)
31. Höller, D., Behnke, G., Bercher, P., Biundo, S.: Language classification of hierarchical planning problems. In: Proc. of the 21st European Conf. on AI (ECAI), pp. 447–452. IOS Press (2014)
32. Höller, D., Behnke, G., Bercher, P., Biundo, S.: Assessing the expressivity of planning formalisms through the comparison to formal languages. In: Proc. of the 26th Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 158–165. AAAI Press (2016)
33. Höller, D., Bercher, P., Richter, F., Schiller, M., Geier, T., Biundo, S.: Finding user-friendly linearizations of partially ordered plans. In: 28th PuK Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK) (2014)
34. Honold, F., Bercher, P., Richter, F., Nothdurft, F., Geier, T., Barth, R., Hörnle, T., Schüssel, F., Reuter, S., Rau, M., Bertrand, G., Seegebarth, B., Kurzok, P., Schattenberg, B., Minker, W., Weber, M., Biundo, S.: Companion-technology: Towards user- and situation-adaptive functionality of technical systems. In: Int. Conf. on Intelligent Environments (IE), pp. 378–381. IEEE (2014). URL http://companion.informatik.uni-ulm.de/ie2014/companion-system.mp4
35. Honold, F., Schüssel, F., Weber, M.: Adaptive probabilistic fission for multimodal systems. In: Proc. of the 24th Australian Computer-Human Interaction Conf. (OzCHI), pp. 222–231. ACM (2012)
36. Kambhampati, S., Mali, A., Srivastava, B.: Hybrid planning for partially hierarchical domains. In: Proc. of the 15th Nat. Conf. on AI (AAAI), pp. 882–888. AAAI Press (1998)
37. Keyder, E., Geffner, H.: Soft goals can be compiled away. Journal of AI Research (JAIR) **36**, 547–556 (2009)
38. Kowalski, R.A.: Predicate logic as programming language. In: IFIP Congress, pp. 569–574 (1974)
39. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: Proc. of the 5th European Semantic Web Conf. (ESWC), pp. 629–643. Springer (2008)
40. Lyons, J.B., Koltai, K.S., Ho, N.T., Johnson, W.B., Smith, D.E., Shively, R.J.: Engineering trust in complex automated systems. Ergonomics in Design **24**(1), 13–17 (2016). DOI 10.1177/1064804615611272
41. McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proc. of the 9th Nat. Conf. on AI (AAAI), pp. 634–639. AAAI Press (1991)
42. Muir, B.M.: Trust in automation: Part I. theoretical issues in the study of trust and human intervention in automated systems. Ergonomics **37**(11), 1905–1922 (1994)
43. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: SHOP2: An HTN planning system. Journal of AI Research (JAIR) **20**, 379–404 (2003)
44. Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Wu, D., Yaman, F., Muñoz-Avila, H., Murdock, J.W.: Applications of SHOP and SHOP2. Intelligent Systems, IEEE **20**, 34–41 (2005)
45. Nebel, B., Koehler, J.: Plan reuse versus plan generation: A theoretical and empirical analysis. Artificial Intelligence **76**(1-2), 427–454 (1995)
46. Nguyen, X., Kambhampati, S.: Reviving partial order planning. In: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI), pp. 459–466. Morgan Kaufmann (2001)

47. Nothdurft, F., Bertrand, G., Heinroth, T., Minker, W.: GEEDI - guards for emotional and explanatory dialogues. In: 6th Int. Conf. on Intelligent Environments (IE), pp. 90–95. IEEE (2010)
48. Nothdurft, F., Honold, F., Zablotskaya, K., Diab, A., Minker, W.: Application of verbal intelligence in dialog systems for multimodal interaction. In: 10th Int. Conf. on Intelligent Environments (IE), pp. 361–364. IEEE (2014)
49. Parasuraman, R., Riley, V.: Humans and automation: Use, misuse, disuse, abuse. Human Factors: The Journal of the Human Factors and Ergonomics Society **39**(2), 230–253 (1997)
50. Penberthy, J.S., Weld, D.S.: UCPOP: A sound, complete, partial order planner for ADL. In: Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR), pp. 103–114. Morgan Kaufmann (1992)
51. Porteous, J., Sebastia, L., Hoffmann, J.: On the extraction, ordering, and usage of landmarks in planning. In: Proc. of the 6th European Conf. on Planning (ECP), pp. 37–48. AAAI Press (2001)
52. Russell, S., Norvig, P.: Artificial Intelligence – A Modern Approach, 1 edn. Prentice-Hall (1994)
53. Seegebarth, B., Müller, F., Schattenberg, B., Biundo, S.: Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In: Proc. of the 22nd Int. Conf. on Automated Planning and Scheduling (ICAPS), pp. 225–233. AAAI Press (2012)
54. Simons, M.: Proof presentation for isabelle. In: Proc. of the 10th Int. Conf. on Theorem Proving in Higher Order Logics (TPHOLs), pp. 259–274. Springer (1997)
55. Sohrabi, S., Baier, J.A., McIlraith, S.A.: HTN planning with preferences. In: Proc. of the 21st Int. Joint Conf. on AI (IJCAI), pp. 1790–1797. AAAI Press (2009)
56. Trac, S., Puzis, Y., Sutcliffe, G.: An interactive derivation viewer. Electronic Notes Theoretical Computer Science **174**(2), 109–123 (2007)
57. Younes, H.L.S., Simmons, R.G.: VHPOP: Versatile heuristic partial order planner. Journal of AI Research (JAIR) **20**, 405–430 (2003)
58. Zhu, L., Givan, R.: Heuristic planning via roadmap deduction. In: IPC-4 Booklet, pp. 64–66 (2004)

The following pages show the publication:

S. Biundo, D. Höller, B. Schattenberg, and P. Bercher. “Companion-Technology: An Overview”. In: *Künstliche Intelligenz* 30.1 (2016). Special Issue on Companion Technologies, pp. 11–20. DOI: 10.1007/s13218-015-0419-3

https://link.springer.com/article/10.1007/s13218-015-0419-3

The final publication is available at link.springer.com

Künstliche Intelligenz: Special Issue on Companion Technologies
The final article is available at http://link.springer.com/article/10.1007/s13218-015-0419-3

Companion-Technology: An Overview

Susanne Biundo · Daniel Höller · Bernd Schattenberg · Pascal Bercher

Received: 10 November 2015 / Accepted: 17 November 2015

Abstract Companion-technology is an emerging field of cross-disciplinary research. It aims at developing technical systems that appear as "Companions" to their users. They serve as co-operative agents assisting in particular tasks or, in a more general sense, even give companionship to humans. Overall, Companion-technology enables technical systems to smartly adapt their services to individual users' current needs, their requests, situation, and emotion. We give an introduction to the field, discuss the most relevant application areas that will benefit from its developments, and review the related research projects.

Keywords Artificial Companions · Companion-systems · Human-Technology Interaction

1 Introduction

Companion-technology denotes a novel, cross-disciplinary field of research that aims at a paradigm shift in human-technology interaction. It is motivated by two fundamental observations. First, technological progress in informatics and the engineering sciences provides us with technical systems and electronic services of continuously increasing complexity and functional "intelligence" in ever shorter innovation cycles. Second, quite often a considerable lack of comfort and convenience in use lets users feel overstrained and hindered from exploiting the offered functionality of these systems and services to its full extent. Companion-technology points the way out of this dissent by providing the means for a smart, adequate, and particularly user-tailored human-technology interaction.

Technical systems that conform to the Companion-paradigm should be able to smartly adapt their functionality to the individual user's requirements and current needs; adequately react to changes of context and changes of the environment that might impair an effective user-system interaction; be sensitive to the user's emotional state and disposition; and conduct helpful and informative dialogs. To this end, the systems need to be provided with advanced cognitive abilities and rich knowledge sources that build the basis for really advanced and effective user support.

So far, the notion of *Companion* together with technical systems has been used in various ways. Most prominently, the EU-funded COMPANIONS project focused on the development of Companion-systems as conversational agents, which give companionship to human users and accompany their owners over a (life-) long period [92]. Another prominent example are *Robotic Companions* as addressed by one of the pilot finalists of the European Flagship initiative "Robot Companions for Citizens". These biologically inspired systems were aimed at supporting humans in their daily activities [8].

Susanne Biundo · Daniel Höller · Pascal Bercher
Ulm University, Institute of Artificial Intelligence, D-89069 Ulm, Germany. E-mail: forename.surname@uni-ulm.de
Bernd Schattenberg
Büro für intelligente Technologie-Beratung, D-88471 Laupheim, Germany. E-mail: mail@berndschattenberg.de

However, up to now a systemized definition of the essence of Companion-technology or companionable systems is still lacking. The first attempt to come up with such a definition was made when establishing the interdisciplinary Transregional Collaborative Research Centre "*Companion*-Technology for Cognitive Technical Systems" [42, 91, 44]. Here, Companion-systems were specified as cognitive technical systems showing particular characteristics: competence, individuality, adaptability, availability, cooperativeness and trustworthiness. A precise definition of these characteristics together with specifications on how these characteristics could be achieved will be given in a forthcoming book [43].

This article presents the current state of the art in research and development towards Companion-technology. In Section 2 we introduce the research areas the contributions of which are essential for the realization of Companion-technology and review the application areas that will significantly benefit from its development. Section 3 gives a comprehensive overview of research projects that address Companion-technology in some way or another. At that, we do not only refer to currently active projects, but give a summary on the complete history of projects related to the field. Finally, we conclude with some remarks on open issues and further developments in Section 4.

2 Research and Application Areas of Companion-Technology

Companion-technology builds upon wide-ranging cognitive abilities of technical systems. The realization and the synergy of those are, since roughly one decade, investigated under the research theme of *cognitive systems* or *cognitive technical systems*. The theme is focused on capabilities such as environment perception, emotion recognition, planning, and learning, and their combination with advanced human-computer interaction. A first survey on cognitive technical systems was published by Vernon et al. [86], whereas Putze and Schultz give a more recent introduction [78]. Furthermore, there was also a special issue on *Cognition for Technical Systems* of the KI journal [35].

2.1 Research Areas

The upcoming research field of Companion-technology is cross-disciplinary in nature. Obviously, major constituents stem from sub-fields of AI. The supply of rich background knowledge together with the ability to reason about this knowledge is a necessary prerequisite for intelligent and user-tailored system behavior. Planning and decision making techniques enable systems to generate courses of action and to reflect and explain the effects of the respective acting. Thereby, depending on the application at hand, the system may either act itself or take the generated course to give recommendations for action to a human user. As soon as sensory input has to be processed in order to recognize environmental conditions, a user's emotional state or his or her disposition, reasoning under uncertainty becomes indispensable. In these cases, often techniques from machine learning and neural information processing are the methods of choice. Finally, natural language processing plays an important role for the exchange of information between the system and its user.

However, in order to design and conduct effective and adequate dialogs between a companionable system and its human user, advanced dialog management and human-computer-interaction techniques have to be employed and integrated with a system's above-mentioned cognitive abilities. They provide the possibility to choose among various dialog strategies and to interact via various media and modalities. With that, they enable a system to show a communication behavior that fits the user's current needs, the context of the application at hand, and the user's emotional state.

The field of affective computing originating from the work of Picard [74] is an essential contributor to Companion-technology as well. Combining aspects from informatics, psychology, and cognitive science, it is concerned with the recognition, processing, and even the simulation of human affects. Being able to dynamically recognize a user's emotional state helps a system to adapt its acting as well as its interaction and intervention strategies in an adequate way, thereby ensuring that a user gets not overburdened and preventing sudden dialog interruptions by the user.

Psychology and in particular the subfield of human factors investigate the interaction between humans and technical systems systematically and on an empirical basis. The results obtained here and the methodologies used are of particular importance for research, development, and evaluation of Companion-technology. Finally, a synergy with the field of neuro-biology results from neuro-imaging during human-machine interaction, which allows to draw conclusions regarding the benefit of particular such dialogs, for example.

2.2 Application Areas

Most technical systems are operated by human users. There is a wide variety of different application areas in which systems of different kind are used that may enrich their functionality and the system's user friendliness by Companion-technology. Here, we give a short overview about some of these application areas.

Robotics. A large area of research and development concerned with providing assistance to human users is the field of *Robotics*. Especially areas of application where robots interact directly with humans (see [54,58,46] for overviews) form an interesting field for the realization of Companion-characteristics. These areas range from (seemingly) smart toys with limited interaction capabilities, over service robots that perform tasks in household or elderly care, to systems in health care that support people with disabilities or in rehabilitation.

The field of domestic service robots (see [77] for an overview) ranges from single robots performing certain household tasks, like vacuum cleaning [55], over smart robotic environments like an assistive kitchen [36], to systems that are designed to support the user in a broad variety of duties [47,34,32].

A robot system that focuses on the social interaction in a bartender domain has been developed by Petrick and Foster [72]. Its behavior is based on AI planning and on its observations, which are based on vision and speech input.

Health and Elderly Care. Robots are also applied in health and elderly care. These robots can be categorized into *rehabilitation* or *therapy robots* and *assistive social robots* [48,67]. The former do not focus on human machine interaction [48] like e.g. intelligent wheelchairs and are thus not further discussed here. The latter category can be divided into robots that support a human in basic activities like eating or getting dressed as well as pet-like robots [48]. Here, the term "Companion" is used sometimes, but often in a literal sense (i.e., they provide companionship) and not like given above. Especially robots that support humans in their daily lives might benefit from Companion-characteristics. An overview of the field is given in [67]. A survey on the effects of robots in health and elderly care can be found in [48,37]. Examples for such robots are the robot seal "Paro" that is designed for psychotherapy [87,88] or the assistant robot described in [75] that supports elderly people by providing guidance and reminders.

There are also less robot-like intelligent devices that support humans in their daily lives. Systems support people suffering from dementia or other cognitive disabilities in activities of daily life, such as washing hands or brushing teeth, e.g. [76,45,71]. Other systems support people with disabilities, e.g. blind people ([51] gives an overview). Although these systems need less social abilities, situation and user adaptive behavior might help to optimize their capabilities.

Intelligent Environments and Homes. Other application areas that aim at realizing customized assistance are those of intelligent environments [31] and in particular smart homes [59]. Systems in these areas should tailor their behavior to the current situation as well as to the preferences of the human user [31, p.4]. (Smart) homes often have more than one resident, hence adaptation to the current environment and the user's situation also becomes more complex. Smart homes are of particular interest as an application scenario for elderly care, as they allow to monitor and assist their elderly residents and, e.g., call an ambulance or notify other residents in case of an emergency. An important issue is the question on whether potential residents accept this technology in their daily life (see [52,56]); smart homes have been particularly investigated for patients [80]. For an overview of several smart home applications and smart home research we refer to [64,77,79].

Driver Assistance. Modern vehicles become increasingly smarter with a wide range of possible services [40,66, 85]. In particular, we encounter a growing individualization of cars to its user. For example, they arrange the position of the driver's seat or adjust the radio's volume to the preferred level; they are equipped with many sensors that allow to automatically react to the environment (e.g. to break automatically) or to assist parking [89]. Individualization to a user's personal preferences and his current situation will also become important for autonomously driving cars, as one could imagine that the taken tour or speed depends on a combination of a user's personal preferences and his current stress level or emotional state.

Other Assistance Systems. Individualization to a specific user and its environment as well as the capability to behave rationally are important capabilities for smart *assistant systems* in various application fields such as elderly care. Due to its capability to automatically solve complex tasks, AI planning can also serve as the basis to improve the basic functionality of many technical systems, such as smartphones (see e.g. [41]). AI planning also serves as the basis in a prototypical system that provides automated assistance in the task of setting up a home entertainment system [38,39]. The system autonomously calculates a plan that solves the task and presents it as a sequence of detailed instructions that explain which cable needs to be plugged into which port of the respective device. Further, the system can generate AI plan explanations that - transformed into natural language - explain to the user the necessity of any presented instruction in question. The presentation of these instructions can also be adapted to different skill levels and preferences of different users [63,62].

Glodek et al. describe an intelligent ticket vending machine as an example application for explaining how information from various sources, such as sensors and a knowledge base, can be fused [57]. The ticket machine makes use of various sensors to observe the current situation and the user's emotional state in order to adapt its behavior accordingly. For instance, the system monitors whether the user is interacting with the system or with other people in front of the vending machine or talking with someone else using a cell phone. In such a situations, any spoken text or performed gestures are not interpreted as input to the system.

User assistance is also essential in any situation that involves great risks and/or where the cognitive load of the respective user is quite high. One such system, *ELP (Emergency Landing Planner)*, assists pilots in case of an emergency taking into account various factors like weather conditions, for example [69].

3 Research Projects in the Area of Companion-Technology

In this section, we focus on projects that are related in terms of the application's adaptation aspects and facets that aim at *advanced user assistance*, which means in particular: personal assistance systems, care of the elderly, and assistance for impaired or disabled persons. This is of course a very pragmatic and relatively fuzzy categorization of the enormous variety of approaches behind the presented research projects – it is thus a mere entry point to the field for the inclined reader.

3.1 History of Companion-Technology-like Projects

ACCESS (Assisted Cognition in Community, Employment and Support Settings) [2,65] was one of the first major joint research projects to address the issue of cognitive technical systems. Computer scientists and medical researchers at the University of Washington explored the impact of cognitive support for people suffering early stages of Alzheimer's disease. They combined techniques from Artificial Intelligence and Ubiquitous Computing in order to monitor the behavior of the patients and from that to assess their cognitive capabilities. While this research focused on establishing cognitive support, service robotics projects like *Nursebot* [25,82] at the University of Pittsburgh, Carnegie Mellon University, and University of Michigan, or *GiraffPlus* [17,50], funded by the European Community's Framework Programme Seven (FP7), aimed at directly assisting elderly people with cognitive tasks, for example helping them to remember medication schedules. The assistance functionality is thereby not embedded in a user's environment, but provided by an autonomous mobile platform that interacts directly with the patient. But robot Companions have not only been successfully deployed in elderly care scenarios, they are also very well and unbiased received by children, for example within the scope of the FP7 project *ALIZ-E* [4,68] on adaptive strategies for sustainable long-term social interaction. The autonomous system thereby assists in diabetes management during a long-term interaction with the child patient.

The goal of the Collaborative Research Centre (CRC) 588 *Humanoid Robots - Learning and Cooperating Multimodal Robots* [18,53], funded by the German Research Foundation (DFG) at the Karlsruhe Institute of Technology (KIT), was to develop concepts, methods, and mechatronic components for creating humanoid robots. While assisting in household environments, the robot was supposed to grasp and acknowledge its human user's intentions in a natural, human-like, and multi-modal manner. The project had a specific interest in interactive learning, so in cooperatively interacting with a human user, the robot learned from that person new vocabulary, relevant objects in the household environment, and the execution of grasping and manipulation tasks. The research efforts thus focused on a broad variety of techniques like learning from observations, spontaneous speech recognition, and the like.

The relevance of robotic Companions as a means for assisted living on the European level became apparent by the selection of the *Robot Companions for Citizens initiative CA-RoboCom* [8] as one of the seven Future and Emerging Technologies (FET) Flagship Pilot finalists in 2012. The initiative's consortium aimed at developing biologically inspired robots that support humans in their daily activities and in particular in potentially dangerous situations. In order to provide this kind of support, their agenda also included developing adequate cognitive capabilities and addressing emotional aspects of companionship. CA-RoboCom proposed the adoption of autonomous systems on various levels of scale and organization, eventually realizing ubiquitous robotic assistance.

Establishing a new level of usability by providing Companion-like assistance in the realm of information technology was the goal of the following two German national "lead-project" initiatives, funded by the Federal Ministry of Education and Research (BMBF). *SmartKom* [30,90] followed the vision of automatically tailoring interactions with systems to the specific needs of the individual users. The barriers for novices in adopting information technology were to be minimized by providing more self-explanatory, respectively self explaining user interfaces. Key elements therefore were highly adaptive dialog-based interfaces, which combined natural speech, facial expression, gesture, and conventional graphical interfaces. In this way, the systems allowed for addressing all human senses in quasi-natural communication settings. The *EMBASSI project* [16,60,61] focused on assisting the operation of complex technical devices of everyday life, thereby providing access to their functionalities without any deeper technical knowledge required. This was achieved by establishing a flexible conversational-like dialog between the human user and the technical system, with the interfaces being based on psychological and ergonomic studies. In following this direction, the initiative developed intelligent assistance and anthropomorphic interfaces, combined with a corresponding architecture, infrastructure components, and protocol standards.

Focused on multi-modal dialog systems, the *SEMAINE (Sustained emotionally colored machine-human interaction using non-verbal expression)* FP7 initiative [29,83] followed the vision to provide these systems with a technology such that a human user is able to engage with them in an everyday conversation. The key element is to build an avatar with a rich facial expression, who reacts in particular on the user's non-verbal signals and traits of his or her emotional state. The communication, although no "real" content is understood, becomes believably emotionally colored. This "Sensitive Artificial Listener" is the underlying metaphor for SEMAINE's human-computer interface design.

Affective computing became a larger and more general research topic in the area of computer science and so it did in particular in realizing Companion-like functionality. In the mid-2000's, an FP6 project was funded that was exclusively dedicated to research on emotions in the course of man-computer interaction: the *Human-Machine Interaction Network on Emotion (HUMAINE)* [73]. Its members aimed at so-called *Emotion-oriented Systems*, that means, systems that are able to recognize emotions of human users, which can build an adequate representation of the underlying emotional states and processes, and that are capable of interacting on them. This includes research topics on emotion theory, signals, emotional aspects of interaction, emotions in cognition and actions, emotions in communications, and usability aspects of emotion-oriented systems.

After the network's completion, the HUMAINE consortium became the nucleus for *The Association for the Advancement of Affective Computing (AAAC)* [1], which is an active stakeholder in the field. Among others, the AAAC organizes the *Audio/Visual Emotion Challenge and Workshop*, the competition on emotion analysis methods on multi-modal corpora.

In 2006, the DFG established the excellence cluster *Cognition for Technical Systems (CoTeSys)* [15,33,49]. It was devoted to research in the areas of design, implementation, and analysis of information processing methods that constitute cognitive processes in technical systems [33]. In this long-term, interdisciplinary initiative technical systems have been modeled on the human brain with respect to learning and reliably performing complex activities, to adapt to changes in the environment and objectives, and the like. The project joined research in technical disciplines together with neurobiology and cognitive sciences in order to realize and evaluate technical solutions to perception in multimodal sensor feeds, to learning and knowledge acquisition, to behavior planning, organization and control, and to human-computer interaction models.

One of the most influential projects to the notion of Companion-technology is *COMPANIONS* [13, 92], funded under the FP6 programme. It depicted Companion-systems as virtual conversational agents, which communicate with their users primarily via spoken language, but also employ touch-sensitive displays and others sensors. The research agenda of this project included many of the before-mentioned capabilities that are associated with verbal interaction: dialog management, speech recognition and synthesis, and emotion detection and elicitation. In addition to these technical issues, COMPANIONS also examined long-term aspects in the user-Companion relationship like the expanding engagement of users and the increasing demand for respecting their preferences and inclinations. With envisioning Companions to participate in everyday activities, the project also covered their philosophical and social implications. All these topics presented themselves to be even more relevant in application areas like fitness and health coaching or elderly care scenarios.

3.2 Currently Active Projects in Focus Areas

The majority of current projects that deal with pieces of Companion-technology employs the notion of an embodied, more or less anthropomorphic companionship, which in turn often translates into projects on autonomous robotic platforms to assist human users in their everyday lives. We will therefore start by focusing on a few representatives of this direction, which subsumes a vast number of methods and practical solutions.

Robotic companions become more and more socially engaging and in order to build or maintain a believable relationship, this requires them to adapt to the experiences they share with their human users. This personalization of autonomous platforms, together with all its technical and psychological issues is addressed by a number of cognitive robotics projects, for example the *MIT Personal Robots* projects [22] at the Massachusetts Institute of Technology or the researchers training initiative *Applications of Personal Robotics for Interaction and Learning (APRIL)* [5], funded under the Horizon 2020 Framework of the European Union (H2020). Their research centers around adaptation and learning mechanisms in technical systems, with which users prefer to interact in a natural, human-like fashion.

The H2020 project RAMCIP [28] addresses a specific kind of domestic service robot, a *"Robotic Assistant for MCI patients at home"*. The vision is to provide the autonomous system with higher-level cognitive capabilities such that it can pro-actively assist elderly people, patients suffering symptoms of beginning Alzheimer's disease, and the like. In these application domains, the right proportion of a level of discreteness and actual assistance provisioning is key, because the patients' autonomy must not be violated and the persons are supposed to be stimulated properly in order to stay as active as possible.

The aspect of the robotic Companion stimulating its user for medical purposes is also central to *Cognitive Development for Friendly Robots and Rehabilitation (CODEFROR)* [10]: The autonomous system is supposed to provide training and rehabilitation techniques for children with sensory, motor, and cognitive disabilities. It is another example of a target user group that heavily depends on natural and intuitive interfaces. In order to deliver adequate assistance concepts and mechanisms, this FP7 project focuses in particular on developmental issues of human cognition like the evolution of action representations, intentions, and emotions.

PAL, the *Personal Assistant for healthy Lifestyle* [26], builds upon some of the results of *ALIZ-E* (see above). It proposes the combination of a robotic platform, a virtual avatar agent, and a number of interactive health-related devices, which together constitute a health-related assistance system for young patients and their caregivers. Type 1 Diabetes Mellitus is a complex illness with serious risks, which on the one hand requires the patients to acquire and strictly adhere to specific habits within their diabetes regimen and which on the other hand demands for personalized and context-sensitive support in order to reduce the diabetes-associated risks persistently. It is the H2020 project's vision, that the robot and avatar serve as incarnations of the child's personal Companion, while the other system components support their parents and caregivers with respect to information sharing, regimen coordination and the like.

A primarily virtual avatar-based approach is followed by FP7's *Miraculous-Life for Elderly Independent Living* [20], which aims at unobtrusively supporting elderly people in their daily activities and safety needs. The avatar metaphor is chosen to allow the users to connect emotionally more easily. This is supported by providing the avatar with the capacity for behavioral and emotional understanding, allowing for interactions involving, e.g., facial expressions, gestures, and contextual information. This human-like company is expected to stimulate and motivate older people to stay active.

A universal "ease of use" and intuitive interfaces for technical systems in general is the goal for the DFG excellence cluster *Cognitive Interaction Technology (CITEC)* [9,81]. To this end, the researchers examine a variety of cognitive processes concerning interaction and communication, ranging from the integration of perception and motoric functions to mediation mechanisms for shared attention between human users and technical systems. They regard any communication with the systems as situational acts that require to coordinate speech, perception, and motoric action. As a consequence, learning and knowledge acquisition techniques in these areas become central research issues, as well as recognition, analysis, and goal-driven control of attention on objects in the environment as part of an emotional and social interaction.

One important aspect of Companion-technology is to provide systems with the means to enable and process a broad variety of input and output modalities. In this regard we find a large number of HCI-related projects that put emphasis on the sensory side. The *MIT Responsive Environments* laboratory [23] produces examples for augmenting the environment with a plethora of individual sensors and sensor networks. Building a coherent model of integrated sensory information is an ongoing challenge in this area. But the classical sensors for gathering visual and auditory information are not the only concerns of the community: *"Sensory Experiences for Interactive Technologies"* [70], a project funded by the European Research Council (ERC), aims for example at extending interactive technologies by integrating touch, taste, and even smell experiences. And although exploiting gestures as an input for systems is suggested by several research initiatives, the *BODY-UI (Body-based User Interfaces)* ERC project [7] studies the use of the body as a modality for input and output matters likewise. The aim is to understand how a user's cognitive processes are reflected by his or her body, and vice versa, and eventually how this can be exploited for creating natural interfaces. A *Corpus-Based Multimodal Approach to the Pragmatic Competence of the Elderly (CorpAGEst)* [14], funded under FP7, explores the use of speech and gesture modalities in particular amongst elderly people. The interaction patterns and their modalities are supposed to change with age and therefore this project's results may provide exploitable data for adapting assisting systems appropriately. On the subconscious level, a related topic is addressed by *Symbiotic Mind Computer Interaction for Information Seeking (MindSee)* [19] (FP7), which analyses EEG and discreet peripheral physiological sensors and combines them with available interaction context information. The project aims at exploiting these implicit cues of the user's perception and emotions in information retrieval applications. Similar topics are addressed from a different angle by the *ARIA-VALUSPA (Artificial Retrieval of Information Assistants - Virtual Agents with Linguistic Understanding, Social skills, and Personalized Aspects)* H2020 project [6]: The assisting avatar is to be realized as an anthropomorphic character that is capable of holding human-like multi-modal social interactions. Verbal and non-verbal cues are used to modify searches or filter results.

There are many more research initiatives in the field of human-computer interaction, which contribute to the ideas of accessibility in Companion-technology. For instance, the technical implementation of intelligent and reliable multimodal user interfaces is the driving force behind the DFG excellence cluster *Multimodal Computing and Interaction (MMCI)* [24,84]. This includes processing of natural spoken language, dialog management, image processing for three-dimensionally reconstructing scenes and poses, and the synthesis of virtual scenes and interacting avatars. But it also involves technical aspects for providing a solid and trustworthy system infrastructure.

Further examples of the many different facets of interaction issues are the questions that arise from building up a shared understanding of complex contexts from basic concepts when *Communicating with Computers* [11], a program funded by the US Defense Advanced Research Projects Agency. Its vision follows the notion of human users being involved in a symmetric communication with a system when collaboratively developing solutions to given problems, including the language to actually communicate about that problem. Of course, the area of affective computing contributes by bridging the gap between human emotions and information technology - from the large number of projects we refer to the *MIT Affective Computing group* [21] and the before-mentioned *AAAC* [1]. These initiatives provide methods to sense and elicit emotional user states and are in particular suitable for detecting and handling enjoyable, stressful, or otherwise particularly meaningful episodes during an interaction. With a focus on providing natural access to assisting technology over a variety of devices, the *AIDE (Adaptive Multimodal Interfaces to Assist Disabled People in Daily Activities)* H2020 project [3] targets the needs of impaired persons. Here, the challenge lies in a shared-control paradigm for assistive devices, ranging from wearables to pervasive installations. As one final facet in our overview, we mention the European Union's FP7 *Prosperity 4All* initiative [27], which deals with the need for establishing an adequate ecosystem that enables developers to economically build self-personalizing interfaces and their corresponding hardware.

The issues of a universal notion of Companion-technology are comprehensively dealt with by the Transregional Collaborative Research Centre *Companion-Technology for Cognitive Technical Systems* [12,42,91,44,43]. This interdisciplinary initiative, funded by the DFG, systematically investigates cognitive capabilities and their implementation in technical systems. This is done while focusing on a set of key characteristics such as individuality, adaptability, availability, cooperativeness, and trustworthiness. Realizing these so-called Companion-characteristics by the integration of various cognitive processes in technical systems is intended to open a new dimension regarding human-technology interaction. Since the resulting Companion-systems will provide their technical functionality by taking into account the entire current and past situation of the user, including his or her emotional state as well as environmental conditions, these systems will finally be perceived and accepted as competent and empathetic assistants.

4 Conclusion

Companion-technology is an exciting field of research at the interfaces between AI and informatics, the engineering sciences, and the life sciences. It imposes a number of challenges on the disciplines involved and requires close cross-disciplinary co-operation. Among the main issues to be addressed are a consistent interlocking of information processing procedures to actually connect the sensory input levels of a prospective companionable system to its logical planning and decision making levels and vice versa; the design and conduction of comprehensive empirical studies to carefully explore both users' demands on the interaction and dialog capabilities of future technical systems and the effects of these capabilities once they are actually implemented; and finally the development of procedure models and tool support to enable the take-up of the technology in business and industry.

Acknowledgements This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG). We want to thank our colleagues Felix Richter, Thomas Geier, Frank Honold, Georg Layher, and Florian Nothdurft for discussions and pointers to literature.

Project Webpages and References

1. AAAC (the association for the advancement of affective computing). http://emotion-research.net
2. ACCESS (assisted cognition in community, employment and support settings). http://cognitivetech.washington.edu
3. AIDE (adaptive multimodal interfaces to assist disabled people in daily activities). http://aideproject.eu
4. ALIZ-E. http://aliz-e.org
5. APRIL (applications of personal robotics for interaction and learning). http://cordis.europa.eu/project/rcn/197987_en.html
6. ARIA-VALUSPA (artificial retrieval of information assistants - virtual agents with linguistic understanding, social skills, and personalized aspects). http://aria-agent.eu
7. BODY-UI (body-based user interfaces). http://www.body-ui.eu
8. CA-RoboCom (robot companions for citizens). http://www.robotcompanions.eu
9. CITEC (cognitive interaction technology). http://www.cit-ec.de
10. CODEFROR (cognitive development for friendly robots and rehabilitation). https://www.codefror.eu
11. Communicating with computers. http://www.darpa.mil/program/communicating-with-computers
12. Companion-technology for cognitive technical systems. http://www.companion-technology.org
13. COMPANIONS. http://www.companions-project.org
14. CorpAGEst (corpus-based multimodal approach to the pragmatic competence of the elderly). http://corpagest.org
15. CoTeSys (cognition for technical systems). http://cotesys.in.tum.de
16. EMBASSI (Elektronische multimediale Bedien- und Service-Assistenz). http://ftb-esv.de/embass.html
17. GiraffPlus. http://www.giraffplus.eu
18. Humanoid robots - learning and cooperating multimodal robots (CRC 588). http://csl.anthropomatik.kit.edu/english/sfb.php

19. MindSee (symbiotic mind computer interaction for information seeking). http://mindsee.eu
20. Miraculous-life for elderly independent living. http://www.miraculous-life.eu
21. MIT affective computing group. http://www.media.mit.edu/research/groups/affective-computing
22. MIT personal robots project. http://www.media.mit.edu/research/groups/personal-robots
23. MIT responsive environments laboratory. http://www.media.mit.edu/research/groups/responsive-environments
24. MMCI (multimodal computing and interaction). http://www.mmci.uni-saarland.de
25. Nursebot. http://www.cs.cmu.edu/~flo/
26. PAL (personal assistant for healthy lifestyle). http://www.pal4u.eu
27. Prosperity 4all. http://www.prosperity4all.eu
28. RAMCIP (robotic assistant for mci patients at home). http://ramcip-project.eu
29. SEMAINE (sustained emotionally coloured machine-human interaction using non-verbal expression). http://www.semaine-project.eu
30. SmartKom. http://www.smartkom.org
31. Augusto, J.C., Callaghan, V., Cook, D., Kameas, A., Satoh, I.: Intelligent environments: a manifesto. Human-Centric Computing and Information Sciences **3**(1), 1–18 (2013). DOI 10.1186/2192-1962-3-12
32. Awaad, I., Kraetzschmar, G.K., Hertzberg, J.: The role of functional affordances in socializing robots. International Journal of Social Robotics **7**(4), 421–438 (2015). DOI 10.1007/s12369-015-0281-3
33. Beetz, M., Buss, M., Wollherr, D.: Cognitive technical systems – what is the role of artificial intelligence? In: Proc. of the 30th German Conference on Artificial Intelligence (KI), pp. 19–42. Springer (2007). DOI 10.1007/978-3-540-74565-5_3
34. Beetz, M., Jain, D., Mösenlechner, L., Tenorth, M., Kunze, L., Blodow, N., Pangercic, D.: Cognition-enabled autonomous robot control for the realization of home chore task intelligence. Proc. of the IEEE **100**(8), 2454–2471 (2012). DOI 10.1109/JPROC.2012.2200552
35. Beetz, M., Kirsch, A. (eds.): Künstliche Intelligenz – Special Issue on Cognition for Technical Systems, vol. 24, issue 4 (2010)
36. Beetz, M., Stulp, F., Radig, B., Bandouch, J., Blodow, N., Dolha, M., Fedrizzi, A., Jain, D., Klank, U., Kresse, I., Maldonado, A., Marton, Z., Mösenlechner, L., Ruiz, F., Rusu, R.B., Tenorth, M.: The assistive kitchen – a demonstration scenario for cognitive technical systems. In: The 17th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 1–8. IEEE (2008). DOI 10.1109/ROMAN.2008.4600634
37. Bemelmans, R., Gelderblom, G.J., Jonker, P., De Witte, L.: Socially assistive robots in elderly care: A systematic review into effects and effectiveness. Journal of the American Medical Directors Association **13**(2), 114–120 (2012). DOI 10.1016/j.jamda.2010.10.002
38. Bercher, P., Biundo, S., Geier, T., Hoernle, T., Nothdurft, F., Richter, F., Schattenberg, B.: Plan, repair, execute, explain - how planning helps to assemble your home theater. In: Proc. of the 24th International Conference on Automated Planning and Scheduling (ICAPS), pp. 386–394. AAAI Press (2014)
39. Bercher, P., Richter, F., Hörnle, T., Geier, T., Höller, D., Behnke, G., Nothdurft, F., Honold, F., Minker, W., Weber, M., Biundo, S.: A planning-based assistance system for setting up a home theater. In: Proc. of the 29th National Conference on Artificial Intelligence (AAAI), pp. 4264–4265. AAAI Press (2015)
40. Bishop, R.: A survey of intelligent vehicle applications worldwide. In: Proc. of the IEEE Intelligent Vehicles Symposium, pp. 25–30. IEEE (2000). DOI 10.1109/IVS.2000.898313
41. Biundo, S., Bercher, P., Geier, T., Müller, F., Schattenberg, B.: Advanced user assistance based on AI planning. Cognitive Systems Research **12**(3-4), 219–236 (2011). DOI 10.1016/j.cogsys.2010.12.005. Special Issue on Complex Cognition
42. Biundo, S., Wendemuth, A.: Von kognitiven technischen Systemen zu *Companion*-Systemen. Künstliche Intelligenz **24**(4), 335–339 (2010). DOI 10.1007/s13218-010-0056-9
43. Biundo, S., Wendemuth, A. (eds.): Companion Technology – A Paradigm Shift in Human-Technology Interaction. Springer (2016). Forthcoming
44. Biundo, S., Wendemuth, A.: *Companion*-technology for cognitive technical systems. Künstliche Intelligenz (2016). DOI 10.1007/s13218-015-0414-8. Special Issue on Companion Technologies
45. Boger, J., Hoey, J., Poupart, P., Boutilier, C., Fernie, G., Mihailidis, A.: A planning system based on markov decision processes to guide people with dementia through activities of daily living. IEEE Transactions on Information Technology in Biomedicine **10**(2), 323–333 (2006). DOI 10.1109/TITB.2006.864480
46. Breazeal, C., Takanishi, A., Kobayashi, T.: Social robots that interact with people. In: Springer Handbook of Robotics, pp. 1349–1369 (2008). DOI 10.1007/978-3-540-30301-5_59
47. Breuer, T., Macedo, G.R.G., Hartanto, R., Hochgeschwender, N., Holz, D., Hegger, F., Jin, Z., Mueller, C.A., Paulus, J., Reckhaus, M., Ruiz, J.A.Á., Plöger, P., Kraetzschmar, G.K.: Johnny: An autonomous service robot for domestic environments. Journal of Intelligent and Robotic Systems **66**(1-2), 245–272 (2012). DOI 10.1007/s10846-011-9608-y
48. Broekens, J., Heerink, M., Rosendal, H.: Assistive social robots in elderly care: a review. Gerontechnology **8**(2), 94–103 (2009). DOI 10.4017/gt.2009.08.02.002.00
49. Buss, M., Beetz, M.: CoTeSys – cognition for technical systems. Künstliche Intelligenz **24**(4), 323–327 (2010). DOI 10.1007/s13218-010-0061-z
50. Coradeschi, S., Cesta, A., Cortellessa, G., Coraci, L., Galindo, C., Gonzalez, J., Karlsson, L., Forsberg, A., Frennert, S., Furfari, F., Loutfi, A., Orlandini, A., Palumbo, F., Pecora, F., von Rump, S., Štimec, A., Ullberg, J., Ötslund, B.: Giraffplus: A system for monitoring activities and physiological parameters and promoting social interaction for elderly. In: Human-Computer Systems Interaction: Backgrounds and Applications 3, *Advances in Intelligent Systems and Computing*, vol. 300, pp. 261–271. Springer International Publishing (2014). DOI 10.1007/978-3-319-08491-6_22

51. Dakopoulos, D., Bourbakis, N.G.: Wearable obstacle avoidance electronic travel aids for blind: a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **40**(1), 25–35 (2010). DOI 10.1109/TSMCC.2009.2021255
52. Demiris, G., Rantz, M.J., Aud, M.A., Marek, K.D., Tyrer, H.W., Skubic, M., Hussam, A.A.: Older adults' attitudes towards and perceptions of 'smart home' technologies: a pilot study. Informatics for Health and Social Care **29**(2), 87–94 (2004). DOI 10.1080/14639230410001684387
53. Dillmann, R., Asfour, T.: Collaborative research center on humanoid robots (SFB 588). Künstliche Intelligenz **22**(4), 26–28 (2008)
54. Fong, T., Nourbakhsh, I.R., Dautenhahn, K.: A survey of socially interactive robots. Robotics and Autonomous Systems **42**(3-4), 143–166 (2003). DOI 10.1016/S0921-8890(02)00372-X
55. Forlizzi, J., DiSalvo, C.: Service robots in the domestic environment: a study of the roomba vacuum in the home. pp. 258–265. ACM (2006). DOI 10.1145/1121241.1121286
56. Gaul, S., Ziefle, M.: Smart home technologies: Insights into generation-specific acceptance motives. In: HCI and Usability for e-Inclusion, *Lecture Notes in Computer Science*, vol. 5889, pp. 312–332. Springer (2009). DOI 10.1007/978-3-642-10308-7_22
57. Glodek, M., Honold, F., Geier, T., Krell, G., Nothdurft, F., Reuter, S., Schüssel, F., Hörnle, T., Dietmayer, K., Minker, W., Biundo, S., Weber, M., Palm, G., Schwenker, F.: Fusion paradigms in cognitive technical systems for humancomputer interaction. Neurocomputing **161**, 17–37 (2015). DOI 10.1016/j.neucom.2015.01.076
58. Goodrich, M.A., Schultz, A.C.: Human-robot interaction: A survey. Foundations and Trends in Human-Computer Interaction **1**(3), 203–275 (2007). DOI 10.1561/1100000005
59. Harper, R.: Inside the smart home. Springer Science & Business Media (2006)
60. Herfet, T., Kirste, T., Schnaider, M.: EMBASSI multimodal assistance for infotainment and service infrastructures. Computers & Graphics **25**(4), 581–592 (2001). DOI 10.1016/S0097-8493(01)00086-3
61. Hildebrand, A., Sá, V.: EMBASSI: Electronic multimedia and service assistance. In: Proc. of the Intelligent Interactive Assistance & Mobile Multimedia Computing (IMC) (2000)
62. Honold, F., Bercher, P., Richter, F., Nothdurft, F., Geier, T., Barth, R., Hörnle, T., Schüssel, F., Reuter, S., Rau, M., Bertrand, G., Seegebarth, B., Kurzok, P., Schattenberg, B., Minker, W., Weber, M., Biundo, S.: Companion-technology: Towards user- and situation-adaptive functionality of technical systems. In: 10th International Conference on Intelligent Environments (IE), pp. 378–381. IEEE (2014). DOI 10.1109/IE.2014.60
63. Honold, F., Schüssel, F., Weber, M.: Adaptive probabilistic fission for multimodal systems. In: Proc. of the 24th Australian Computer-Human Interaction Conference, OzCHI '12, pp. 222–231. ACM (2012). DOI 10.1145/2414536.2414575
64. Jiang, L., Liu, D.Y., Yang, B.: Smart home research. In: Proc. of the Third Conference on Machine Learning and Cybernetics, vol. 2, pp. 659–663. IEEE (2004). DOI 10.1109/ICMLC.2004.1382266
65. Kautz, H., Arnstein, L., Borriello, G., Etzioni, O., Fox, D.: An overview of the assisted cognition project. In: AAAI Workshop on Automation as Caregiver: The Role of Intelligent Technology in Elder Care, pp. 60–65 (2002)
66. Lindgren, A., Chen, F.: State of the art analysis: An overview of advanced driver assistance systems (ADAS) and possible human factors issues. In: Human factors and economics aspects on safety – Proc. of the Swedish Human Factors Network (HFN) Conference, pp. 38–50 (2006)
67. Van der Loos, H.F.M., Reinkensmeyer, D.J.: Rehabilitation and health care robotics. In: Springer Handbook of Robotics, pp. 1223–1251 (2008). DOI 10.1007/978-3-540-30301-5_54
68. Aliz-e Project Team: The ALIZ-E project: adaptive strategies for sustainable long-term social interaction. Poster and Demo Track of the 35th German Conference on Artificial Intelligence at KI 2012 (2012)
69. Meuleau, N., Plaunt, C., Smith, D.E., Smith, T.: An emergency landing planner for damaged aircraft. In: Proc. of the 21st Innovative Applications of Artificial Intelligence Conference (IAAI), pp. 114–121. AAAI Press (2009)
70. Obrist, M., Tuch, A.N., Hornbaek, K.: Opportunities for odor: Experiences with smell and implications for technology. In: Proc. of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2843–2852. ACM (2014). DOI 10.1145/2556288.2557008
71. Peters, C., Hermann, T., Wachsmuth, S., Hoey, J.: Automatic task assistance for people with cognitive disabilities in brushing teeth - a user study with the tebra system. ACM Transactions on Accessible Computing (TACCESS) **5**(4), 1–34 (2014). DOI 10.1145/2579700
72. Petrick, R., Foster, M.E.: Planning for social interaction in a robot bartender domain. In: Proc. of the 23rd International Conference on Automated Planning and Scheduling (ICAPS), pp. 389–397. AAAI Press (2013)
73. Petta, P., Pelachaud, C., Cowie, R. (eds.): Emotion-Oriented Systems: The Humaine Handbook. Cognitive Technologies. Springer (2011). DOI 10.1007/978-3-642-15184-2
74. Picard, R.W.: Affective Computing, vol. 252. MIT press Cambridge (1997)
75. Pineau, J., Montemerlo, M., Pollack, M., Roy, N., Thrun, S.: Towards robotic assistants in nursing homes: Challenges and results. Robotics and Autonomous Systems **42**(3), 271–281 (2003)
76. Pollack, M.E.: Planning technology for intelligent cognitive orthotics. In: Proc. of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS), pp. 322–332. AAAI Press (2002)
77. Prassler, E., Kosuge, K.: Domestic robotics. In: Springer Handbook of Robotics, pp. 1253–1281 (2008). DOI 10.1007/978-3-540-30301-5_55
78. Putze, F., Schultz, T.: Adaptive cognitive technical systems. Journal of Neuroscience Methods **234**, 108–115 (2014). DOI 10.1016/j.jneumeth.2014.06.029

79. Rashidi, P., Mihailidis, A.: A survey on ambient-assisted living tools for older adults. IEEE Journal of Biomedical and Health Informatics **17**(3), 579–590 (2013). DOI 10.1109/JBHI.2012.2234129
80. Rialle, V., Duchene, F., Noury, N., Bajolle, L., Demongeot, J.: Health "smart" home: Information technology for patients at home. Telemedicine Journal and E-Health **8**(4), 395–409 (2002)
81. Ritter, H.J.: Cognitive interaction technology - goals and perspectives of excellence cluster CITEC. Künstliche Intelligenz **24**(4), 319–322 (2010). DOI 10.1007/s13218-010-0063-x
82. Roy, N., Baltus, G., Fox, D., Gemperle, F., Goetz, J., Hirsch, T., Margaritis, D., Montemerlo, M., Pineau, J., Schulte, J., Thrun, S.: Towards personal service robots for the elderly. In: Workshop on Interactive Robots and Entertainment (WIRE) (2000)
83. Schröder, M., Bevacqua, E., Cowie, R., Eyben, F., Gunes, H., Heylen, D., ter Maat, M., McKeown, G., Pammi, S., Pantic, M., Pelachaud, C., Schuller, B., de Sevin, E., Valstar, M., Wöllmer, M.: Building autonomous sensitive artificial listeners. IEEE Transactions on Affective Computing (TAC) **99**(1), 1–1 (2011)
84. Seidel, H.: Excellence cluster "multimodal computing and interaction" – robust, efficient and intelligent processing of text, speech, visual data, and high dimensional representations. it - Information Technology **50**(4), 253–257 (2008). DOI 10.1524/itit.2008.0492
85. Shaout, A., Colella, D., Awad, S.: Advanced driver assistance systems - past, present and future. In: Seventh International Computer Engineering Conference (ICENCO), pp. 72–82 (2011). DOI 10.1109/ICENCO.2011.6153935
86. Vernon, D., Metta, G., Sandini, G.: A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. Transactions on Evolutionary Computation **11**(2), 151–180 (2007). DOI 10.1109/TEVC.2006.890274
87. Wada, K., Shibata, T.: Social effects of robot therapy in a care house - change of social network of the residents for two months. In: International Conference on Robotics and Automation, pp. 1250–1255. IEEE (2007). DOI 10.1109/ROBOT.2007.363156
88. Wada, K., Shibata, T., Musha, T., Kimura, S.: Robot therapy for elders affected by dementia. IEEE Engineering in Medicine and Biology Magazine **27**(4), 53–60 (2008). DOI 10.1109/MEMB.2008.919496
89. Wada, M., Yoon, K.S., Hashimoto, H.: Development of advanced parking assistance system. IEEE Transactions on Industrial Electronics **50**(1), 4–17 (2003). DOI 10.1109/TIE.2002.807690
90. Wahlster, W. (ed.): SmartKom: Foundations of Multimodal Dialogue Systems. Springer (2006). DOI 10.1007/3-540-36678-4
91. Wendemuth, A., Biundo, S.: A companion technology for cognitive technical systems. In: Cognitive Behavioural Systems, Lecture Notes in Computer Science, pp. 89–103. Springer (2012)
92. Wilks, Y.: Close Engagements with Artificial Companions: Key social, psychological, ethical and design issues, *Natural Language Processing*, vol. 8. John Benjamins Publishing (2010)

Susanne Biundo is a professor of Computer Science at Ulm University and the Director of the DFG-funded Transregional Collaborative Research Centre *Companion-Technology for Cognitive Technical Systems* SFB/TRR 62. She led the *European Network of Excellence in AI Planning* and acted as the Co-chair of various international AI conferences. Susanne Biundo was elected ECCAI Fellow in 2004 and was a founding member of the Executive Council of ICAPS, the *International Conference on Automated Planning and Scheduling*. Her research interests are in AI Planning, Automated Reasoning, Knowledge Modeling, and Cognitive Systems.

Daniel Höller received his M. Sc. in Computer Science from Bonn-Rhein-Sieg University of Applied Sciences in 2013 and joined the Institute of Artificial Intelligence at Ulm University afterwards. He is interested in Automated Planning and currently working on his Ph. D. on Plan and Goal Recognition in the context of the SFB/TRR 62.

Bernd Schattenberg is a freelance developer and consultant for AI planning and scheduling (P&S) technology since 2013. Before that, he was a Research Assistant at the Institute of Artificial Intelligence of Ulm University, where he received his Ph.D. in 2009. He coordinated the *European Network of Excellence in AI Planning* and the Transregional Collaborative Research Centre *A Companion-Technology for Cognitive Technical Systems* SFB/TRR 62. His interests include knowledge engineering methods, mixed-initiative aspects, and novel application areas of hybrid P&S.

Pascal Bercher received his diploma in Computer Science from the University of Freiburg, Germany. He is now working on his Ph.D. in the SFB/TRR 62 at the AI Institute of Ulm University. His area of research focuses on hierarchical task network (HTN) planning, Partial-Order Causal-Link (POCL) planning, and the fusion thereof, called hybrid planning. His thesis is concerned with theoretical foundations, planning heuristics, and practical applications of these planning paradigms.

The following pages show the publication:

P. Bercher, F. Richter, T. Hörnle, T. Geier, D. Höller, G. Behnke, F. Nothdurft, F. Honold, W. Minker, M. Weber, and S. Biundo. “A Planning-based Assistance System for Setting Up a Home Theater”. In: *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI 2015).* AAAI Press, 2015, pp. 4264–4265

Reprinted with kind permission of AAAI Press.

A Planning-based Assistance System for Setting Up a Home Theater

Pascal Bercher and **Felix Richter** and **Thilo Hörnle** and **Thomas Geier** and
Daniel Höller and **Gregor Behnke** and **Florian Nothdurft** and **Frank Honold** and
Wolfgang Minker and **Michael Weber** and **Susanne Biundo**
Faculty of Engineering and Computer Science
Ulm University, Germany
email: forename.surname@uni-ulm.de

Abstract

Modern technical devices are often too complex for many users to be able to use them to their full extent. Based on planning technology, we are able to provide advanced user assistance for operating technical devices. We present a system that assists a human user in setting up a complex home theater consisting of several HiFi devices. For a human user, the task is rather challenging due to a large number of different ports of the devices and the variety of available cables. The system supports the user by giving detailed instructions how to assemble the theater. Its performance is based on advanced user-centered planning capabilities including the generation, repair, and explanation of plans.

Introduction

Technical devices become more and more complex and often cause mental overload to human users operating them. Companion Technology (Biundo and Wendemuth 2010; Wendemuth and Biundo 2012) enables the development of companion systems – cognitive technical systems that assist the user in operating a technical device. In our paper *"Plan, Repair, Execute, Explain - How Planning Helps to Assemble your Home Theater"* (Bercher et al. 2014), we showed how planning and HCI capabilities were integrated to allow for advanced user assistance – illustrated in an example scenario, where a human user wants to connect various devices of his home theater. Here, we describe a mobile version of the prototypical *Companion System* presented there.

Application Scenario – Assembling a Home Theater

We want to assist a human user in assembling her or his HiFi components, s.t. every component receives the required audio/video signals. We consider an example scenario, where the user wants to set up a theater consisting of a television, a blu-ray player, a satellite receiver, and an audio/video receiver. More technically, the task is to connect these devices in such a way that the television receives the video signals of the blu-ray player and the satellite receiver and that the audio/video receiver receives the audio signals of these devices. For connecting the devices, there are several different cables and adapters available.

We are not aware of any tool that is capable of assisting a user with such an assembly task. In practice, one would have to consider the operating manuals of the various devices and to come up with a solution by oneself.

It is straightforward to model the task in terms of a planning problem, where actions correspond to plugging cables into devices. We described an excerpt of the domain model[1] in our earlier work (Bercher et al. 2014). We solve the planning problem that encodes the assembly task using a hybrid planning approach. It fuses hierarchical planning with Partial-Order Causal Link (POCL) planning. That approach is well-suited for our intent of providing user assistance, as the causal links allow for the explanation of plans (Seegebarth et al. 2012), thereby justifying the system's behavior. Further, they allow for the smooth integration of plan execution and repair. The hierarchy, if introduced, can be used for limiting the search space and improving plan explanations.

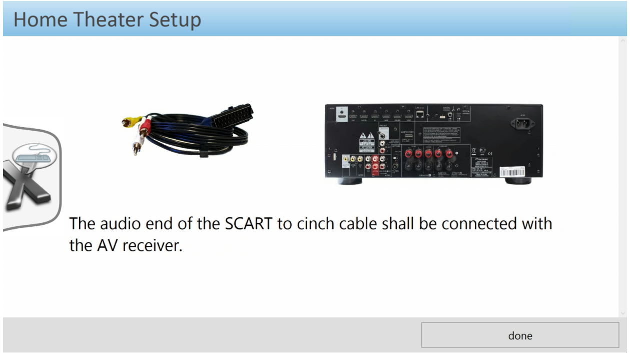
While the generation of plans allows to give detailed instructions to the user in the first place, plan repair allows to assist in cases, where execution failures occur. Plan explanation allows to question the necessity of certain instructions.

System Description

Our system implements a generic architecture for Companion Systems introduced in earlier work (Bercher et al. 2014, Fig. 1). Its domain-independent components enable the realization of assistance systems in many application areas.

In our example application, user assistance is based upon a sequence of instructions that, if executed by the user, solves the assembly task. That instruction sequence is based upon a plan, which is a solution to the given planning problem. Since in hybrid planning such plans are only partially ordered, a most-suitable total order must be chosen. While every total order respecting the partial order of the plan solves the given planning problem, some of them might be more plausible to human users than others. We devel-

[1] The domain model can be downloaded from the SiGAPS website http://users.cecs.anu.edu.au/~patrik/sigaps/


Home Theater Setup

The audio end of the SCART to cinch cable shall be connected with the AV receiver.

Figure 1: The figure shows a single user instruction that visualizes a single action of a solution plan. The respective ports of the audio/video receiver are flashing in red.

oped several domain-independent plan linearization strategies (Höller et al. 2014) and employed a strategy that seems most plausible in our example scenario.

We now investigate how a single user instruction looks like. Every instruction is based upon a (primitive[2]) action. The corresponding action schema basically looks as follows:

$$\mathit{plugIn}(\textsc{src-H}, \textsc{src-P}, \textsc{snk-H}, \textsc{snk-P})$$

That action schema has four parameters. The terms SRC-H and SNK-H represent the source and the sink hardware, respectively. The terms SRC-P and SNK-P represent the ports (such as HDMI) of the two hardware devices that are used for plugging SRC-H and SNK-H together. When presenting an action, these constants are used by the dialog and interaction management (Honold et al. 2013) to generate an appropriate user interface. This includes pictures of the devices and their ports as well as natural language text that verbalizes the respective instruction (see Fig. 1). We have done an empirical evaluation of our system with test subjects. A majority of these subjects perceived the system very well in particular because of the pictures of the devices and the highlighting of the used ports (Bercher et al. 2014).

During interaction, the user may always state execution errors or ask for justification of the currently presented instruction. For that purpose the user may touch/click on the large X on the left side of the presented instruction (see Fig. 1). Then a dropdown box occurs listing "The cable is broken!" and "Why should I do this?". The user may also interact with the system using speech input that is recognized using off-the-shelf software.

If "The cable is broken!" is selected, the currently used cable is marked as unusable and the system initiates plan repair. After a solution has been found that incorporates the execution failure (in this case, the unexpectedly broken cable), the new plan is presented to the user in the same way the original plan has been presented before. In our demonstration, we did not model unplug actions, so cables that have already been used cannot be plugged out (depending on the cables that became unusable this might be necessary to find a repaired solution, however).

The user may also select "Why should I do this?". In that case the system uses plan explanation to derive a justification for the currently presented action. Such a justification is a chain of proof steps proving the purpose of the respective instruction. That proof is translated into natural language and presented to the user. As an example consider the explanation for the action depicted by Fig. 1: "You have to connect the SCART to cinch cable to the AV receiver to transmit audio data from the satellite receiver to the AV receiver."

Discussion & Future Work

We described a system that supports a human user in the task of setting up her or his home theater. The system's capabilities include plan generation, plan execution, plan repair, and plan explanation. The task to solve is encoded as a planning problem given in advance. To obtain a fully general system, we want to enable to user to specify the given hardware and the task to solve. We also want to extend the domain model to allow unplugging cables. Concerning plan repair, our demo system only allows to specify broken cables as execution failure. However, our plan repair approach is more general and allows to handle arbitrary state changes. So, we want to allow the user to specify any state variable that unexpectedly changed its truth value.

Acknowledgment

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

Bercher, P.; Biundo, S.; Geier, T.; Hoernle, T.; Nothdurft, F.; Richter, F.; and Schattenberg, B. 2014. Plan, repair, execute, explain - how planning helps to assemble your home theater. In *Proc. of ICAPS 2014*, 386–394. AAAI Press.

Biundo, S., and Wendemuth, A. 2010. Von kognitiven technischen Systemen zu Companion-Systemen. *Künstliche Intelligenz* 24(4):335–339.

Höller, D.; Bercher, P.; Richter, F.; Schiller, M.; Geier, T.; and Biundo, S. 2014. Finding user-friendly linearizations of partially ordered plans. In *28th PuK Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK 2014).*

Honold, F.; Schüssel, F.; Weber, M.; Nothdurft, F.; Bertrand, G.; and Minker, W. 2013. Context models for adaptive dialogs and multimodal interaction. In *Proc. of the 2013 9th Int. Conf. on Intelligent Environments (IE 2013)*, 57–64.

Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In *Proc. of ICAPS 2012*, 225–233. AAAI Press.

Wendemuth, A., and Biundo, S. 2012. A companion technology for cognitive technical systems. In *Proc. of the EUCogII-SSPNET-COST2102 Int. Conf. (2011)*, Lecture Notes in Computer Science, 89–103.

[2] In hybrid planning, actions may be primitive or abstract. Solution plans only contain primitive actions, however.

The following pages show the publication:

P. Bercher, S. Biundo, T. Geier, T. Hörnle, F. Nothdurft, F. Richter, and B. Schattenberg. “Plan, Repair, Execute, Explain - How Planning Helps to Assemble your Home Theater”. In: *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014).* AAAI Press, 2014, pp. 386–394

Reprinted with kind permission of AAAI Press.

The included PDF is a revised version. Modifications are not explicitly mentioned, since there were only minor corrections or improvements.

Plan, Repair, Execute, Explain – How Planning Helps to Assemble your Home Theater

Pascal Bercher, Susanne Biundo, Thomas Geier
Thilo Hoernle, Felix Richter, Bernd Schattenberg
Institute of Artificial Intelligence,
University of Ulm, Germany,
email: firstName.lastName@uni-ulm.de

Florian Nothdurft
Institute of Communications Engineering
University of Ulm, Germany
email: florian.nothdurft@uni-ulm.de

Abstract

In various social, work-related, or educational contexts, an increasing demand for intelligent assistance systems can be observed. In this paper, we present a domain-independent approach that combines a number of planning and interaction components to realize advanced user assistance. Based on a hybrid planning formalism, the components provide facilities including the generation, execution, and repair as well as the presentation and explanation of plans. We demonstrate the feasibility of our approach by means of a system that aims to assist users in the assembly of their home theater. An empirical evaluation shows the benefit of such a supportive system, in particular for persons with a lack of domain expertise.

Introduction

Today's rapid technological progress periodically provides us with technical systems, services and devices of increasingly complex, "intelligent", functionality. Those systems include household appliances, web services, cars, service robots, electronic devices such as smartphones, and much more. However, it is not always easy to operate these systems and, even less so, to actually exploit the whole range of their potential. In many cases, users are overwhelmed or stressed out by awkward operation menus, inappropriate interaction modes, or inconvenient or even missing instruction manuals. Moreover, the demand for assistance systems that support users in their every-day life by helping them to accomplish certain tasks or to handle systems and devices appropriately becomes more and more important, in particular in view of the aging society.

We show how AI planning technology can be employed to perform user assistance in a smart and valuable fashion. We introduce a system architecture composed of a number of planning components, a knowledge base, as well as components to manage the system's dialog and interaction with the user. The planning components include a hybrid planner that combines hierarchical concepts with those of partial-order causal-link planning. In addition, a plan execution system, as well as plan repair and explanation components are parts of the architecture. User assistance is based on automatically generated plans. Such a plan of actions is passed to the user in a stepwise manner. The user's execution of these actions is monitored. This way, the system is able to detect execution failures or deviations from the proposed plan and to initiate the repair of the original plan accordingly. Furthermore, explanations can be given, which elucidate why a certain action is part of the plan, for example. Plans and explanations need to be communicated to users in an adequate manner, however. Therefore, the system's facilities of establishing dialogs with users and interacting through various modalities are controlled by advanced dialog and interaction management components. They enable the system to give instructions and explanations via displayed text and speech, as well as to understand users' spoken responses and input via touch screen. We used the planning and interaction components and their orchestration within the overall architecture to implement a prototype system that assists users in the assembly of their home theater. Finally, we carried out an empirical user study to evaluate the acceptance and benefit of such an assistance system. It showed that the system's support was perceived very well, in particular by non-experts.

Planning technology has been recently used in several human computer interaction-related contexts. One line of research aims at supporting persons with various impairments, such as children with autism (Bernardini and Porayska-Pomsta 2013) or persons suffering from dementia (Boger et al. 2005). Another line investigates systems that realize assistance functionality by performing certain interaction-related tasks presently done by humans, such as household chores (Beetz et al. 2012) or serving drinks (Petrick and Foster 2013). These approaches do not incorporate facilities for revising or explaining the decisions of the system, however. Approaches that include some form of plan repair exist, e.g., the O-Plan architecture (Tate, Drabble, and Kirby 1994), or in the context of interactive storytelling (Porteous, Cavazza, and Charles 2010) or robotics (Lemai and Ingrand 2004).

In the next section, we present the system architecture and the interplay of its major constituents. The following sections describe these constituents in more detail, focusing on the planning components. After that, we present the user study together with its results and conclude the paper.

System Architecture

We have developed an architecture which integrates essential planning capabilities (plan generation, execution/monitoring, repair, explanation) with components to provide advanced human-computer interaction (dialog and interaction management). The architecture and the domain-independent components enable the implementation of concrete assistance systems in a variety of application areas. Such systems are capable of multi-modally interacting with a human user and to provide intelligent assistance based on the various capabilities provided by the planning and interaction components. The architecture is depicted in Fig. 2. We will shortly explain how the different components interact, before we describe their particular functionality in more detail.

Assistance functionality is provided through the interplay of various planning capabilities; the most basic one being that the system can give a course of actions to a user which solves his current problem at hand. We assume that these problems are formalized in terms of a hybrid planning domain. Knowledge about the specific domain is stored in the *Knowledge Base component*, which manages and processes the information necessary for the other components to work.

As soon as a user requests assistance in terms of a course of action to solve a given problem, the Knowledge Base component passes the corresponding hybrid planning problem to the *Plan Generation component* (arrow 1 in Fig. 2). That component generates a solution, i.e., a plan, and passes it to both the *Plan Execution component* and the *Plan Explanation component* (arrow 2). The latter is done to be prepared in case the user wants to ask questions about the current plan. The plan is executed step by step, and several components (arrows 3 to 10) are involved for each such step.

The Plan Execution component identifies the plan step to be executed next and sends it to the *Dialog Management component* (arrow 3). This way, it can be presented to the user. The component loads the dialog model associated with the plan step and calculates the most-appropriate *dialog goals* to achieve the effects of the given plan step. The chosen dialog goals are passed one by one to the *Interaction Management component* (arrow 4), which builds the actual user interface to be presented to the user (arrow 5). The input of the user is sent back over the Interaction Management component to the Dialog Management component (arrow 8). This constitutes both explicit input entered via the user interface (arrow 6), and implicit input (e.g., facial expression), which is registered by the sensors (arrow 7).

Once a dialog goal is finished, its effects are sent to the Knowledge Base component (arrow 9). A common pattern is to associate a dialog goal to a plan step, and interpret the user's input as an acknowledgement of having achieved the effects of the plan step. The effects registered by the Interaction Management component are then combined with sensor inputs within the Knowledge Base component to calculate a belief about the current world state. This information is then passed back to the Plan Execution component (arrow 10). That component compares the believed state with the expected state, using the effects of the last plan step. If they match, the next plan step is executed in the same way. If a deviation from the expected state is detected, the Plan Execution component triggers the *Plan Repair component* (arrow 11). Repair is initiated by constructing an *augmented planning problem*, which incorporates the found discrepancies together with the execution state of the currently enacted plan. The augmented problem is then sent to the Plan Generation component (arrow 12), which finds a new solution replacing the old plan, s.t. a new cycle can start over.

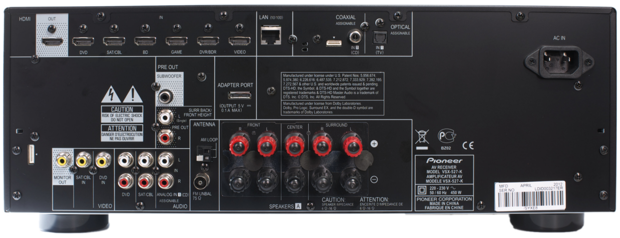


Figure 1: The figure shows the back panel of the amplifier used in our domain model and user study.

At any point in the interaction, the user may ask why he had to execute any of the presented plan steps. If he does so, the Dialog Management component requests the explanation of the current plan step from the Plan Explanation component (arrow 13), which generates a formal explanation stating why this plan step is necessary to solve the overall problem. This explanation is sent back to the Dialog Management component, which initiates its presentation to the user (arrows 14 and 4 to 8).

To illustrate how such an architecture works in detail, we implemented a prototypical system that assists a person in the task of setting up a complex home theater. The interaction between the various components is realized with the message-oriented middleware Semaine (Schröder 2010). The main physical features of the prototype are a multi-device user interface (one touch-screen, one remote terminal) and user-localization based on a laser-range-finder (Honold et al. 2013, Figure 10). No additional sensory modules were used, in particular there were no means of emotion recognition, or means of automatically recognizing the state of the home theater setup and cables. Progress on the task was solely reported through the dialog system.

The task of the user is to connect several devices of his new home theater. They comprise a satellite receiver, a blu-ray player, an amplifier, and a television. Finally, the user wants to be able to use both the blu-ray player and the satellite receiver. While this task may seem easy for an expert, it is quite complicated for non-experts: Fig. 1 shows the back panel of the modeled amplifier (an audio/video receiver, which we also used in the evaluation) and illustrates the high number of different ports and thus the difficulty of the task. In addition to these devices, we modeled several different cables, adapters, and adapter cables (such as a HDMI-to-DVI adapter cable, for example). The information about available devices, cables, and other information relevant for the task (e.g., about the user) is stored in the Knowledge Base component and sent to the respective components if requested.

Knowledge Base

Besides storing all the domain specific information, the Knowledge Base component serves as the central hub for

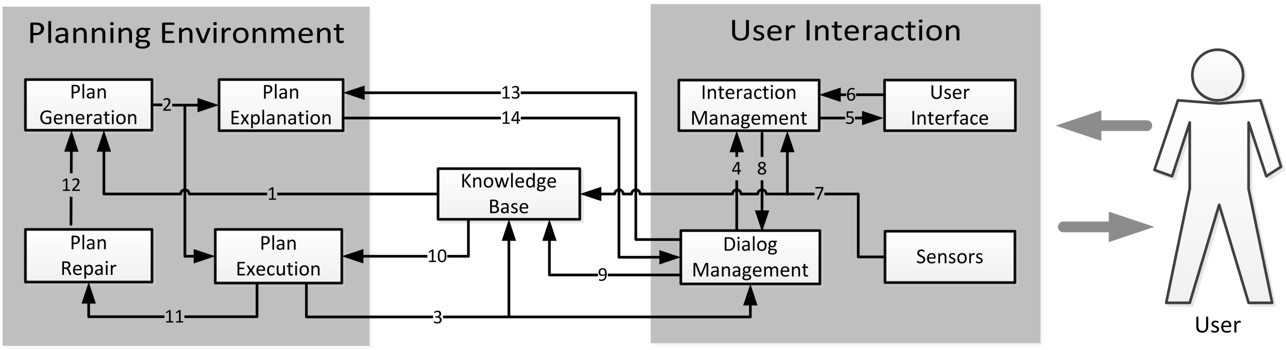


Figure 2: This figure shows the components of our architecture and how they interact with the user and with each other. It is based on the one presented by Honold, Schüssel, and Weber (2012), with emphasis on the planning part instead of the user interaction part. The numbers represent the actual data sent from one component to another: 1: planning problem, 2: generated solution plan, 3: next plan step, 4: next dialog goal, 5: interaction output, 6: explicit interaction input, 7: implicit sensor input, 8: fusion result, 9: interaction effects, 10: actual world state, 11: repair request, 12: plan repair problem, 13: explanation request, 14: formal plan explanation.

our architecture. It integrates information from various sources like sensor data (arrow 7, usually pre-processed by machine learning modules, not depicted), and explicit user input via the Dialog Management component (arrow 9). A major part of this information is of an uncertain nature due to sensor noise, partial observability or model imperfection.

The requirement of a consistent treatment of uncertainty throughout the whole system is addressed by choosing a probabilistic approach. The knowledge of the system is reflected in a probability distribution over the possible world states. Advances in the field over the past 20 years have led us to a point where the treatment of probability distributions over many random variables slowly becomes feasible, e.g., by the use of probabilistic graphical models (Koller and Friedman 2009). To facilitate the concise representation of large graphical models, and in order to ease the integration with other modules like automated planning, we use a logic-like, probabilistic, first-order modeling language called Markov Logic (Richardson and Domingos 2006).

The integration of the knowledge base with the planning components mainly focuses on the provision of a world state. Initially, the Knowledge Base component is queried for all state-features relevant to the current planning problem (arrow 1). In the case of the assembly task, this encompasses knowledge about available devices and cables. During plan execution, relevant state features are then constantly monitored (arrow 10).

To make the probabilistic view of the knowledge base compatible with the deterministic state representation of the planner, one could infer the most probable world state according to the current beliefs and report it as *actual* world state. But for models with many random variables it is very likely that the most probable state still has a vanishingly small probability, and the chance that the true world state differs from the reported one is huge. To handle this, we declare the part of the global model that comes from the planning domain as quasi-deterministic. In this part, upon each new time-step, variables that have not received an explicit observation (e.g., from the dialog) are automatically observed to the most probable state they had during the last time step. This makes the quasi-deterministic part behave like a database, where values only change upon a write request, offering semantics that are compatible with assumptions of deterministic planning. Still, the probabilistic part can transparently use information from the quasi-deterministic part for inferences. The usefulness of such a connection has been demonstrated by its exploitation for improved user identification (Geier et al. 2012).

Plan Generation

In order to facilitate the provision of a suitable plan, the Plan Generation component queries the current world state from the Knowledge Base component. A valid plan will then consist of a course of actions, whose execution will transform the current world state into a desired state. For the assembly task, actions are usually of the form "Connect port A of cable X to port B of device Y.", and the goal conditions consist of the transmission of certain signals (audio or video) from source devices to their respective sink devices.

In our system, such tasks are modeled by means of a hybrid planning problem (Biundo and Schattenberg 2001; Biundo et al. 2011; Geier and Bercher 2011). Hybrid planning combines hierarchical planning (Erol, Hendler, and Nau 1994; Marthi, Russell, and Wolfe 2008) with concepts known from Partial-Order Causal-Link (POCL) planning (McAllester and Rosenblitt 1991; Younes and Simmons 2003). Both paradigms seem to be well-suited for our enterprise of assisting human users: The hierarchical aspect of the domain captures the hierarchical structure many real-life problems show and the explicit representation of causality using causal links allows for the explanation of plans (Seegebarth et al. 2012). These explanations may further benefit from the hierarchical structure, as it allows more flexibility w.r.t. the level of detail of explanations. We believe that the capability of systems to explain their own behavior is essential when assisting human users, since explanations can improve the trust a user has in the system and might lead to higher acceptance rates (Lim, Dey, and Avrahami 2009).

In hybrid planning, the task is modeled by means of a planning domain $\mathcal{D}$ and a problem instance $\mathcal{I}$. For our example application, the planning domain models the specific technical devices (ports male/female, audio/video/either/both signal in/out, etc.) and how they can be connected using the available cables. The problem instance specifies the actual task to solve. In hybrid planning, actions may be either primitive or abstract. In both cases, an action $a(\bar{\tau})$ has the form $\langle pre, \mathit{eff} \rangle$ with *pre* and *eff* being conjunctions of literals over the action parameters $\bar{\tau} = \tau_1, \ldots, \tau_n$. Preconditions specify in which states an action is applicable, and effects of actions specify how a state changes if the action is applied. States and action applicability and application are defined as usual. A partial plan is a structure $\langle PS, \prec, VC, CL \rangle$ consisting of a set of plan steps *PS*, which are uniquely identified actions, either primitive or abstract. The partial order between these plan steps is achieved by the relation $\prec$. The set *VC* contains the variable constraints, which co- or non-codesignate action parameters to other parameters or constants. We call an action ground if all of its parameters are codesignated to some constant taking the closure of *VC*. A plan is called ground if all its actions are ground. The set *CL* contains all causal links of the partial plan P. A causal link $ps \rightarrow_{\varphi} ps'$ denotes that the precondition literal φ of plan step ps' is supported (protected) by the plan step ps. While primitive actions can be executed if all their preconditions are supported by causal links, abstract actions must be further refined, even if their preconditions are all supported. To that end, the domain $\mathcal{D}$ contains at least one so-called *decomposition method* for every abstract action. A decomposition method $m = \langle a(\bar{\tau}), VC_m, P \rangle$ maps an abstract action $a(\bar{\tau})$ to a partial plan P, which “implements” the preconditions and effects of $a(\bar{\tau})$ (Biundo and Schattenberg 2001). Applying a method to a partial plan results in removing the abstract action and replacing it by the partial plan the method maps to, adding the variable constraints VC_m, which relate the action parameters to the variables in P, and to pass on the causal links of the abstract action to its new sub-actions.

Now, the domain $\mathcal{D} = \langle A, M \rangle$ contains the set of primitive and abstract actions A as well as the set of decomposition methods M. The problem instance $\mathcal{I}$ is given in terms of an initial partial plan P_{init}, which contains some abstract actions representing the high-level specification of the tasks to be achieved. It also contains two artificial actions *init* and *goal*, which encode the initial state and the goal description, respectively. The action *init* has no precondition and a complete description of the initial state as effect, including the facts initially false in terms of negative literals[1]. The action *goal* has the goal description as precondition, which is a conjunction of (possibly negative) literals. The solution to a planning problem is a plan P, s.t.:

1. P is a refinement of P_{init}.
2. P does not contain abstract actions.
3. P does not show unprotected precondition literals.
4. P has no causal threats. A causal threat is the situation where the ordering constraints allow a plan step ps'' with effect φ to be ordered between two plan steps ps and ps' which share a causal link $ps \rightarrow_{\varphi'} ps'$, s.t. there is a unification σ with $\varphi = \neg\sigma(\varphi')$.

Criterion 1 is essential for relating any solution to the initial plan P_{init}. In contrast to PDDL, where the problem is given merely in terms of a goal description, the goals in hierarchical/hybrid planning are given in terms of P_{init}. Criterion 2 ensures that only primitive actions are present, as only those can be executed directly. Criteria 3 and 4 together ensure that every plan step linearization which is consistent with P's ordering constraints is an applicable action sequence and generates a state which satisfies the goal description.

Domain Model. We will now give an overview over how we modeled the problem of setting up a home theater. In our planning domain, the initial state specifies the available hardware and its compatibility to other devices. We differentiate between devices and cables, which both have an individual sort DEVICE and CABLE, respectively. Sorts correspond to the concept of types in PDDL. For every device and cable in the scenario, there is a constant of the respective sort, such as BLURAY of sort DEVICE to model the blu-ray player. We also introduce an abstract sort HARDWARE with sub sorts DEVICE and CABLE to be used by the actions. Every hardware in the scenario has several ports: Cables, for instance, ordinarily have two ports, one for each end of the cable (however, more complicated cables can have more than one port at the same end). We thus introduce a sort PORT. Every port of a specific hardware has certain properties: it may be either a *signal in or out*, either *used or unused*, either *male or female*, and it may be used for *video, audio, both* (in case of HDMI), *or either* (for instance, a cinch video cable may be used for either audio or video, but not for both at the same time as is the case for HDMI). The description of the hardware also specifies that certain devices are sources of audio and video signals. For instance, while the TV and the amplifier initially do not have any signal, the blu-ray player and the satellite receiver initially have both video as well as an audio signal. In our domain, we therefore have a predicate HASVIDEO(HARDWARE,DEVICE) expressing that the constant of sort HARDWARE has the video signal produced by the constant of sort DEVICE. In our scenario, the initial state thus contains the atoms HASVIDEO(BLURAY,BLURAY) and HASVIDEO(RECEIVER,RECEIVER) as well as the respective counter parts for the audio signal.

Actions are the means to connect cables to devices. More precisely, we modeled the actions in such a way that they can only be applied if the two ports to be connected are both unused and compatible with each other (w.r.t. gender, for instance). Furthermore, actions may only be applied in the direction of the signals in order to propagate that signal to the cable just plugged in. That is, a cable with an input port may only be connected to a cable or device with an output port if the other cable/device already has some signal. Then,

[1] In POCL planning, we need to specify the facts initially false to be able to support negative preconditions of actions.

the cable that has been plugged in will also have that signal. Note that we did not model *unplug* actions. Thus, in case of execution failures, our prototype might not always find a repaired solution in certain situations.

Hierarchy may be introduced to reduce the search space by specifying pre-defined solutions to sub-problems. For example, an abstract action CONNECT(BLURAY,AMPLIFIER) can be modeled using several decomposition methods, each corresponding to a valid solution transporting both audio and video from the blu-ray player to the amplifier. The initial plan may then contain that abstract action as well as CONNECT(RECEIVER,AMPLIFIER) and CONNECT(AMPLIFIER, TV).

Search Procedure. We search for solutions via search in the space of plans by step-wise refining the initial partial plan until a solution has been found. We base our search technique on the standard POCL planning paradigm (Younes and Simmons 2003), in which plan elements violating solution criteria induce so-called flaws. We extend this technique by being able to decompose abstract actions; we call the resulting system PANDA (Biundo and Schattenberg 2001; Elkawkagy et al. 2012). The search is a two-stage process. First, a most-promising plan is selected from a set of candidates. That selection is done using informed heuristic functions. Our system supports pure non-hierarchical heuristics (Nguyen and Kambhampati 2001; Younes and Simmons 2003; Bercher, Geier, and Biundo 2013; Bercher et al. 2013), as well as hierarchical ones (Elkawkagy et al. 2012). After a most-promising plan has been selected, one of its flaws is selected to be resolved. Resolving a flaw generates a set of successor plans, which can contain new flaws. For example, the insertion of an action adds one flaw for each of its precondition literals. That procedure is repeated until a solution has been created.

Plan Execution and Monitoring

Given a solution plan, the Plan Execution component executes it in a step by step way. When prompted to execute the next plan step, it first determines the set of executable plan steps. A plan step is executable if it has not yet been executed, whereas all plan steps which are ordered before it, have already been executed. The Plan Execution component then chooses a plan step to execute from this set. While any execution order will guarantee that the goal will be reached, some orders can be more intuitive for the user. The implemented system therefore employs a domain-specific approach to execute semantically connected tasks in direct succession, if possible. For example, several steps are needed to transport a video signal from the satellite receiver to the TV. If the last executed step was one of these steps, the component prefers executing another such step next. In future work, we would like to examine the extent to which this could be done domain-independently, for example by maximizing the number of common parameter values of two consecutively ordered tasks.

The chosen plan step is passed on to the Interaction Management component, where the actual execution is performed. Once the execution is completed, the interaction management passes the results of the execution to the Knowledge Base component and sends a signal to the Plan Execution component. The Plan Execution component then checks whether the action had the intended effects. This is done by comparing the expected state – as computed by applying the specified effects of the action to the last known world state – with the actual world state obtained by querying the knowledge base. When the expected and actual world states match, the next plan step is executed.

However, it might happen that the actual world state deviates from the expected state. For example, when plugging an HDMI cable into the blu-ray player fails because the HDMI cable is broken, the actual world state will be that the cable is unusable and that it is not connected to the blu-ray player. In this case, it has to be checked whether the new world state interferes with the yet-to-be-executed parts of the plan. This can be done by examining the presently *active* causal links of the plan, i.e., the causal links whose producer task has been executed while its consumer task has not.

If there is no active causal link for a given literal then it is either irrelevant for executing the remainder of the plan or a yet-to-be-executed task will reestablish it at a later point. Therefore, if the current state agrees with the expected state on all literals for which there is an active causal link, the plan is sure to still be a valid solution. Otherwise, the plan contains tasks (that might be executed far in the future) whose preconditions are no longer supported by valid causal links and the plan needs to be repaired.

Plan Repair

When the Plan Execution component detects that the plan at hand cannot be executed as intended, that plan, together with the problems detected during execution, is passed on to the Plan Repair component, which generates a "repair problem". The Plan Generation component uses this repair problem to generate a repaired solution, if possible.

There might be several reasons for a plan to fail. Often, the reason lies in the non-deterministic and partially observable nature of the environment. For example, an action might not be applicable (now or in the future), because one of its precondition literals does not hold at the current world state while the system predicted that precondition to hold. In our domain, for instance, a cable might be defective or a device port might not be working against expectation.

In these cases we need to find another – working – solution to the original planning problem, which can cope with the unpredicted changes of the world. If the new problem turns out to be unsolvable, we at least need to inform the user that it is no longer possible to set up the home theater system with the remaining set of cables and adapters. While in standard state-based planning, *replanning* seems most attractive (Nebel and Koehler 1995), it is not that easy in hierarchical approaches like we follow, since solutions need not only be executable, but also be a refinement of the initial plan (cf. solution criterion 1). We thereby follow a *plan repair* approach, which is suited for this additional constraint (Bidot, Biundo, and Schattenberg 2008; Biundo et al. 2011).

The repair mechanism basically works as follows: Given the planning domain $\mathcal{D} = \langle A, M \rangle$, the initial problem instance $\mathcal{I}$ given by P_{init}, and the failed solution plan P, the Plan Execution component creates a repair problem instance $\mathcal{I} = \langle P_{init}, O \rangle$, which now contains a set of *obligations* O. This set contains an existence obligation for all plan steps which have already been executed. Satisfying these obligations will guarantee that executed plan steps are part of any new solution. This is important, as we require solutions which are refinements of the initial plan and the plan steps already executed may be essential to satisfy that criterion. Since these plan steps have been executed in one specific order (while the plan that was to be executed might only be *partially* ordered), O also contains ordering obligations which restrict these plan steps to be totally ordered and prevent any "new" action to be ordered within this sequence. Lastly, O contains an obligation that requires a so-called *process* to be inserted right behind the executed plan step sequence. A process is a primitive action with no precondition and the unpredicted world changes as effect.

The planning procedure is altered in such a way that unsatisfied obligations are represented as additional flaws. Then, plans with no flaws are solutions to the original planning problem, they contain the already executed plan steps, and can cope with the unforeseen changes caused by the environment.

Plan Explanation

So, for example, when the system learns about the fact that one of the cables is broken, it comes up with a new solution. This solution is usually more complicated than the original one, since the problem of a missing cable must be worked around. A broken cable could be addressed by using adapters and adapter cables as a replacement. However, the pro-active generation of a new plan, in particular a more complicated one, may confuse the user. Even worse, the system can produce changes in the plan that are not anticipated by the user, because the deviation was not detected via user interaction (e.g., the user reporting the failure of the cable), but it was detected via different means, e.g., computer vision. The explanation of such unexpected or otherwise opaque decisions is critical for the human-computer interaction, because especially unexpected or not understandable situations may have a negative impact on the human-computer trust relationship (Muir 1992). Studies have shown that if the user does not trust the system, the interaction may suffer. This includes reduced frequency or way of interaction, and in the worst case the complete abortion of future interaction (Parasuraman and Riley 1997). Of course, as we want technical systems to become intelligent assistants and help us in complex, as well as in critical situations, it is clear that this should be impeded.

In human-human interaction moments of unclear, not reasonable decisions by one party are often clarified by explaining the process of reasoning (i.e., increasing transparency and understandability). Analogous to that, research by Lim et al. (2009) showed that different kinds of explanations can improve the trust in and the understandability of context-aware intelligent systems. The results showed that *Why* and *Why-not* explanations were the best kind of explanation to increase the user's understanding in the system, though trust was only increased by providing *Why* explanations.

Therefore, our planning system can help improve trust and understandability by providing, if requested, plan explanations. The special case of plan explanation used in this setting is the task of explaining why a given plan step is present in a given plan. To generate explanations, an axiomatic system Σ comprising formalizations of basic arguments about a plan and their logical consequences is constructed (Seegebarth et al. 2012). The axioms are derived from the solution criteria, the plan at hand, and the problem specification. We use the atom $N(ps)$ for representing that a step ps is necessary for a given plan P to constitute a solution. For a formula ϕ encoding a reason that supports the presence of ps in P, Σ contains an axiom of the form $\forall ps.[\phi \Rightarrow N(ps)]$. Constructing an explanation for the presence of a plan step is equivalent to finding a sequence of "applications" of these axioms.

The reason for the presence of a plan step ps in a plan P, according to the solution criteria for hybrid plans, can be one of the following: First, ps is trivially either $init$, $goal$, or any plan step from the problem specification, i.e., Σ contains $N(init)$, $N(goal)$, and $N(ps)$ for all ps in P_{init}. Second, ps establishes a precondition of some other plan step ps'. For this, Σ contains an atom $CR(ps, \varphi, ps')$ for every causal link $ps \rightarrow_{\varphi} ps'$ present in P, representing the *causal relation* between ps and ps'. The necessity of further plan steps can be recursively derived from causal relations to necessary plan steps with the following axiom:

$$\forall ps.[[\exists ps', \varphi.[CR(ps, \varphi, ps') \land N(ps')]] \Rightarrow N(ps)] \quad (1)$$

The third reason for the presence of a plan step ps in P is that ps has been introduced by decomposing an abstract plan step ps' during the construction of P. In contrast to causal dependencies, this information is not represented explicitly in the solution plan and has to be (re-)constructed from the knowledge about the plan generation process: for each decomposition of an abstract plan step ps' via a method m, performed during the construction of P from P_{init}, Σ contains an atom $DR(ps, m, ps')$ for each plan step ps introduced by m, representing the *decomposition relation* between ps and ps'. Plan step necessity can again be recursively derived:

$$\forall ps.[[\exists ps', m.[DR(ps, m, ps') \land N(ps')]] \Rightarrow N(ps)] \quad (2)$$

The result of an explanation request is a formal proof that has to be conveyed to the user in a more suitable from, for instance verbally. The simplest way of achieving this is to provide a text template for each axiom and translating an explanation proof step by proof step.

As an example, consider the case where a video signal needs to be transported from the blu-ray player to the TV, reflected in the goal description via HASVIDEO(TV,BLURAY). The solution plan contains a sequence of plan steps connecting devices and cables. The tasks are connected with causal links successively providing the HASVIDEO(X,BLURAY) property to the next task in the sequence. If the user requests an explanation for the necessity of the plan step plugging a cable into the blu-ray player, the Plan Explanation component uses the above axioms to successively determine the need to derive the necessity of all plan steps in the sequence, including the goal step. Since the goal step is necessary by definition, so are all the other plan steps in the sequence.

Explanations subsume several kinds of explanations like *Why*, *Why-not*, *How-to*, or *What*. Plan explanations in our system are meant to increase the understandability in the system's decisions by using *Why* explanations. Therefore, plan explanation is to be distinguished from explanation of declarative knowledge, which uses *What* and *How-to* explanations to impart knowledge (Nothdurft and Minker 2012).

Dialog and Interaction Management

In the case of required user-interaction, plan steps are transmitted to the Dialog Management component (cf. Fig. 2). Here, the provided plan step is decomposed into a hierarchical dialog structure which consists of so-called dialog goals (Nothdurft et al. 2010). Each dialog goal represents a single interaction step between human and technical system (e.g., one screen on a device filled with specific information). The term dialog "goal" arises from the fact that every step in the interaction pursues a goal. This goal is, in this case, to achieve one or several of the desired plan step effects. Therefore, the term *dialog goal* is to be distinguished from the term *goal* used in planning.

This means on the one hand that a plan step may be decomposed into several steps of interaction, and on the other hand that for every desired plan step effect a set of similar dialog goals may exist. These similar dialog goals may for example take into account user knowledge or emotions, and therefore differ in the complexity and kind of content presentation. The selection of the next dialog goal is made in a user-adaptive manner and leads to an individual dialog appropriate for the current user (Honold et al. 2013). For information presentation the dialog goal is passed on to the Interaction Management component (cf. Fig. 2) and by that transferred to a XML-based format called *Dialog Output*.

Hereby, the dialog content is composed of multiple information objects referencing so-called *information IDs* in the information model. Each information ID can consist of different types (e.g., text, audio, and pictures) which are selected and combined at runtime by a fission subcomponent to compose the user interface in a user- and situation-adaptive way. The reasoning process about the most-appropriate user interface is based on a set of evaluation functions. These functions rate the selection of possible output modalities and combinations by a reward function and propagate the best one for presentation (Honold, Schüssel, and Weber 2012).

Performed user input is perceived by the input devices and transmitted to the multimodal fusion (Schüssel, Honold, and Weber 2013). The interaction input is passed on back to the Dialog Management component, which analyzes how the interaction result influences the desired plan step effects. The effects of the user interaction are transferred to the Knowledge Base component and the Plan Execution component is notified that the current plan step has been processed by the dialog management.

Additionally to that, if the user requested an explanation why the current plan step has to be executed, the dialog management can transfer a plan explanation request to the Plan Explanation component.

Experimental Evaluation

We conducted an experiment to evaluate the benefit of providing plan explanation for the acceptance of an automated assistive system. The experiment is designed as a controlled, randomized trial. This allows us to investigate the effects of plan explanations in a principled way. For reasons of reproducibility, we recreated a fixed course from the prototype system as a seemingly interactive HTML5 slide show. Participants are confronted with the task of connecting several devices of a home theater system consisting of a satellite receiver, a blu-ray player, an amplifier, and a television, as explained in the beginning of the paper. Fig. 1 shows the back panel of the amplifier used in the study. The requirements to the solution were that both watching satellite, TV, and blu-ray disks is possible. The task was complicated by the fact that the enacted solution uses an adapter for one connection. Participants were given an electronic assistant that gave them instructions on which cable to connect to which device through written and spoken text, as well as by images of the devices with the concerned ports highlighted. Participants were randomly, and without them knowing, assigned into two groups. The control group ($n = 29$) was faced with the task as described. The treatment group ($n = 30$) was shown two plan explanations at fixed steps. The longer explanation was: "This step served the goal to transmit the video signal of the blu-ray player to the TV. To this end, the video signal of the blu-ray player is transmitted over the HDMI-to-DVI adapter and the HDMI-to-DVI cable to the amplifier. From there it is transmitted over the video-cinch cable to the TV.".

The outcome of the trial was captured by a self-report questionnaire administered right after the task. Most importantly we asked for the subjectively perceived certainty that the way how the devices are connected fulfills the stated requirements as a 5-point Likert question (e.g., five levels from "No Chance" to "Certain"). In addition to demographic variables, we asked about expertise and prior experience with similar tasks, several other questions about how the participant perceived the device, and questions about how the explanations were perceived (only for the treatment group).

Our main hypothesis was that the plan explanations have a positive effect on the subjectively perceived certainty in the validity of the setup solution.

We recorded 59 subjects in total. Concerning demographic variables, we have at 3 female and 7 male subjects aged over 30, and 19 female and 27 male subjects aged at most thirty; age is missing for three subjects; 26 subjects had a university degree. Nine participants alone were Ph.D. students in chemistry. Only 7 subjects' graduation was lesser than high school equivalent. In general the education level of the sample was overly high. Only 12 participants had a technical background (computer science, engineering); though this does not include natural sciences and Mathematics, to which another 18 subjects affiliated.

We constructed a summary variable capturing the overall perception of the system by summing up all questions that rate aspects of the system (trust, patronization, appeal, utility, etc.; with good internal consistency $\alpha = 0.83$). According to this scale, the system was very well perceived in general (mean/sd: $26.63/3.67$ points out of 30). We found a major influence on this rating in the answer on a question (5-point Likert scale) asking people to judge their confidence to do the setup as required, but while only using the device manuals. The answer to that question significantly predicted the overall perception in a linear regression ($\beta = -0.37, t(55) = -2.99, p < .01$). Thus, people who consider themselves unskilled liked the system better. We also found that women rated the system better than men (mean/sd female $28.14/1.93$, male $25.67/4.19$, significant with $p < .01$ using Mann-Whitney-U, $Z = -2.81$).

Considering the differences between subjects in the treatment group and those in the control group, the main hypothesis that explanation fosters confidence in the implemented solution cannot be supported by the experimental results. In contrary, subjects of the treatment group had lower confidence on average (mean/sd: control $4.66/0.55$, treatment $4.50/0.82$, not significant). However, the confidence of the subjects in the correctness of their assembly was very high in general. Some subjects rated their confidence "certain", even though they forgot to connect some of the cables. Subjects within the treatment group also had lower overall perception scores compared to the control group (mean/sd: control $27.4/2.6$, treatment $25.8/4.4$, not significant).

We attribute the result concerning the explanation feature to several factors. First, there is no substantial risk involved in making a mistake while connecting the components. A simple experiment (turn the devices on and see if it works) can yield a reliable answer without much effort. So being pushed towards reasoning about the correctness may be perceived as annoying. Remember that the explanations were displayed without an explicit request by the user, to facilitate randomization. A second reason for the result within the treatment group is the possibility of an unsettling effect of the explanation, caused by the participants thinking they are to be fooled into believing something wrong — facilitated by the experimental condition.

When looking only at the treatment group, we found that people with a higher education level rated the explanation aspect[2] better (mean/sd: with German "Abitur" $14.9/5.38$, with university degree $17.9/5.43$; not significant).

In addition to the structured part, the questionnaire contained free questions, asking the participants about aspects they particularly liked or disliked. An aspect that was praised in most comments was that we provided photographs of the devices with highlighted ports. The negative counterpart were complaints about the text-to-speech engine used to read the instructions and the explanations, which were also raised by a vast majority of the participants. Comments about the system in general were all positive: "assists in a useful way", "this assistant would be great for my parents.", or "this assistance system is very useful, as it allows people without expertise to follow the instructions successfully". The comments also confirm our assessment that not the explanations themselves were perceived negatively, but the fact that they were mandatory: Some participants mentioned as positive that the explanations were completely optional, as one can simply proceed by clicking on "next" (they were not intended to be optional, but some people proceeded without reading them). Other comments suggested to present the explanation *before* the instruction it refers to, rather than afterwards. There were only five comments about the actual quality of the explanations. Four of them were positive, only one mentioned that explanations were complicated by mentioning the technical names for the cables. Some of the positive remarks were "explanations were good and useful, but the presented version was confusing due to the bad, automated voice" and "the explanations seem to be unnecessary at first glance, but they increase the understanding of what one does and strengthen the credibility of the system".

We conclude that a system that offers automated support is perceived very well, in particular by non-experts. Furthermore, explanation abilities appear to be an important and beneficial feature of assistance systems, while their proactive provision should be thoroughly considered.

Conclusion

We presented a domain-independent architecture to implement plan-based assistance systems. It integrates planning capabilities with advanced dialog and interaction management components, which enable multi-modal communication skills of such systems. The approach offers a high degree of flexibility by involving plan repair mechanisms to assist users in cases where unexpected execution failures occur. Based on the architecture, we implemented a prototype system capable of assisting users in the task of setting up a complex home theater. We demonstrated the acceptance and usefulness of this system in an empirical evaluation.

Acknowledgment

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

Beetz, M.; Jain, D.; Mosenlechner, L.; Tenorth, M.; Kunze, L.; Blodow, N.; and Pangercic, D. 2012. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proc. of the IEEE* 100(8):2454–2471.

Bercher, P.; Geier, T.; Richter, F.; and Biundo, S. 2013. On delete relaxation in partial-order causal-link planning. In *Proc. of the 25th IEEE Intl. Conf. on Tools with AI (ICTAI 2013)*, 674–681.

Bercher, P.; Geier, T.; and Biundo, S. 2013. Using state-based planning heuristics for partial-order causal-link planning. In *Proc. of the 36nd German Conf. on AI (KI 2013)*, 1–12.

[2] sum over 6 questions with good internal consistency of $\alpha = 0.84$

Bernardini, S., and Porayska-Pomsta, K. 2013. Planning-based social partners for children with autism. In *Proc. of the 23rd Intl. Conf. on Automated Planning and Scheduling (ICAPS 2013)*, 362–370.

Bidot, J.; Biundo, S.; and Schattenberg, B. 2008. Plan repair in hybrid planning. In *Proc. of the 31st German Conf. on AI (KI 2008)*, 169–176.

Biundo, S., and Schattenberg, B. 2001. From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *Proc. of the 6th European Conf. on Planning (ECP 2001)*, 157–168.

Biundo, S.; Bercher, P.; Geier, T.; Müller, F.; and Schattenberg, B. 2011. Advanced user assistance based on AI planning. *Cognitive Systems Research* 12(3-4):219–236. Special Issue on Complex Cognition.

Boger, J.; Poupart, P.; Hoey, J.; Boutilier, C.; Fernie, G.; and Mihailidis, A. 2005. A decision-theoretic approach to task assistance for persons with dementia. In *Proc. of the 19th Intl. Joint Conf. on AI (IJCAI 2005)*, 1293–1299.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving hierarchical planning performance by the use of landmarks. In *Proc. of the 26th Natl. Conf. on AI (AAAI 2012)*, 1763–1769.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1994. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proc. of the 2nd Intl. Conf. on AI Planning Systems (AIPS 1994)*, 249–254.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proc. of the 22nd Intl. Joint Conf. on AI (IJCAI 2011)*, 1955–1961.

Geier, T.; Reuter, S.; Dietmayer, K.; and Biundo, S. 2012. Track-person association using a first-order probabilistic model. In *Proc. of the 24th IEEE Intl. Conf. on Tools with AI (ICTAI 2012)*, 844–851.

Honold, F.; Schüssel, F.; Weber, M.; Nothdurft, F.; Bertrand, G.; and Minker, W. 2013. Context models for adaptive dialogs and multimodal interaction. In *Proc. of the 2013 9th Intl. Conf. on Intell. Env.'s (IE 2013)*, 57–64.

Honold, F.; Schüssel, F.; and Weber, M. 2012. Adaptive probabilistic fission for multimodal systems. In *Proc. of the 24th Australian Computer-Human Interaction Conference (OzCHI 2012)*, 222–231.

Koller, D., and Friedman, N. 2009. *Probabilistic Graphical Models: Principles And Techniques*. The MIT Press.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *Proc. of the 19th Natl. Conf. on AI (AAAI 2004)*, 617–622.

Lim, B. Y.; Dey, A. K.; and Avrahami, D. 2009. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proc. of the SIGCHI Conf. on Human Factors in Comp. Systems (CHI 2009)*, 2119–2128.

Marthi, B.; Russell, S. J.; and Wolfe, J. 2008. Angelic hierarchical planning: Optimal and online algorithms. In *Proc. of the 18th Intl. Conf. on Automated Planning and Scheduling (ICAPS 2008)*, 222–231.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. of the 9th Natl. Conf. on AI (AAAI 1991)*, 634–639.

Muir, B. M. 1992. Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems. In *Ergonomics*, 1905–1922.

Nebel, B., and Koehler, J. 1995. Plan reuse versus plan generation: A theoretical and empirical analysis. *AI* 76(1):427–454.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning. In *Proc. of the 17th Intl. Joint Conf. on AI (IJCAI 2001)*, volume 17, 459–466.

Nothdurft, F., and Minker, W. 2012. Using multimodal resources for explanation approaches in technical systems. In *Proc. of the 8th Conf. on Intl. Language Resources and Evaluation (LREC 2012)*, 411–415. European Language Resources Association (ELRA).

Nothdurft, F.; Bertrand, G.; Heinroth, T.; and Minker, W. 2010. GEEDI - Guards for Emotional and Explanatory DIalogues. In *6th Intl. Conf. on Intell. Env.'s (IE 2010)*, 90–95.

Parasuraman, R., and Riley, V. 1997. Humans and automation: Use, misuse, disuse, abuse. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 39(2):230–253.

Petrick, R., and Foster, M. E. 2013. Planning for social interaction in a robot bartender domain. In *Proc. of the 23rd Intl. Conf. on Automated Planning and Scheduling (ICAPS 2013)*, 389–397.

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Trans. Intell. Syst. Tech.* 10:1–10:21.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine Learning* 62(1-2):107–136.

Schröder, M. 2010. The SEMAINE API: towards a standards-based framework for building emotion-oriented systems. *Advances in Humam-Computer Interaction* 2010.

Schüssel, F.; Honold, F.; and Weber, M. 2013. Using the transferable belief model for multimodal input fusion in companion systems. In *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction (MPRSS 2012)*, LNCS/LNAI 7742. 100–115.

Seegebarth, B.; Müller, F.; Schattenberg, B.; and Biundo, S. 2012. Making hybrid plans more clear to human users – a formal approach for generating sound explanations. In *Proc. of the 22nd Intl. Conf. on Automated Planning and Scheduling (ICAPS 2012)*, 225–233.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-plan2: an open architecture for command, planning and control. In *Intell. Scheduling*, 213–239.

Younes, H. L. S., and Simmons, R. G. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of AI Research (JAIR)* 20:405–430.

The following pages show the publication:

S. Biundo, P. Bercher, T. Geier, F. Müller, and B. Schattenberg. “Advanced user assistance based on AI planning”. In: *Cognitive Systems Research* 12.3-4 (Apr. 2011). Special Issue on Complex Cognition, pp. 219–236. DOI: 10.1016/j.cogsys.2010.12.005

Advanced User Assistance Based on AI Planning

Susanne Biundo, Pascal Bercher, Thomas Geier, Felix Müller,
Bernd Schattenberg

⟨firstname⟩.⟨lastname⟩@uni-ulm.de

Institute of Artificial Intelligence, Ulm University, 89069 Ulm, Germany

Abstract

Artificial Intelligence technologies enable the implementation of cognitive systems with advanced planning and reasoning capabilities. This article presents an approach to use hybrid planning – a method that combines reasoning about procedural knowledge and causalities – to provide user-centered assistance.

Based on a completely declarative description of actions, tasks, and solution methods, hybrid planning allows for the generation of knowledge-rich plans of action. The information those plans comprise includes causal dependencies between actions on both abstract and primitive levels as well as information about their hierarchical and temporal relationships.

We present the hybrid planning approach in detail and show its potential by describing the realization of various assistance functionalities based on complex cognitive processes like the generation, repair, and explanation of plans. Advanced user assistance is demonstrated by means of a practical application scenario where an innovative electronic support mechanism helps a user to operate a complex mobile communication device.

Keywords: cognitive technical systems, companion-technology, hybrid planning, plan repair, plan explanation, real-world planning

1. Introduction

In professional and private daily business and especially when pursuing any major undertaking, strategic planning, reasoning about the consequences of acting, and weighing the pros and cons of various options are crucial cognitive capabilities human beings routinely exhibit. When aiming at the construction of advanced intelligent systems that assist users in high-level tasks and support their decision making, it seems therefore quite natural and adequate to rely on a technical equivalent of those cognitive capabilities.

The field of Artificial Intelligence (AI) Planning provides a large variety of methods to plan and reason and to do so by taking specific characteristics of prospective applications and environments into account [1]. Its most popular realm nowadays is classical state-based planning that originates from early work by Fikes and Nilsson [2]. The publication of the *Graphplan* algorithm in the mid-1990s [3] and the setup of the *International Planning Competition* [4] revived the area and launched a strong development towards heuristic forward-search-based planning [5, 6, 7]. While state-based approaches aim at the generation of linear action sequences that are intended to be automatically executed by systems, there are two main strands in the field dedicated to the construction of more elaborate plan structures and to explicitly reflecting the kinds of reasoning humans perform when developing plans. In partial-order causal-link (POCL) planning [8, 9], plans are partially ordered sets of actions and show causal dependencies between actions explicitly. This allows for flexibility w.r.t. the order in which actions are finally executed and enables a human user to grasp the causal structure of the plan and to understand why certain actions are part of it. Hierarchical task network (HTN) planning [10, 11] features another important principle of intelligent planning, namely abstraction. It allows for the specification of both complex abstract tasks and predefined standard solutions for these tasks. Here, plan generation is a top-down refinement process that stepwise replaces abstract tasks by appropriate (abstract) solution plans until an executable action sequence is obtained. HTN planning is particularly useful for solving real-world planning problems since it provides the means to immediately reflect and employ the abstraction hierarchies that are inherent in many domains.

By combining the characteristic features of POCL and HTN techniques *hybrid planning* [12, 13, 14, 15] smoothly integrates reasoning about procedural knowledge and causalities, thereby providing an enabling technology for the realization of complex cognitive capabilities of technical systems. Based on a completely declarative description of actions, tasks, and solution methods, hybrid planning allows for the generation of knowledge-rich plans of action. The information those plans comprise includes causal dependencies between actions on both abstract and primitive levels as well as information about their hierarchical and temporal relationships. By making use of this information, as well as of the underlying declarative domain models, complex cognitive capabilities like the generation of courses of action on various abstraction levels, the stable repair of failed plans in response to some unexpected environment changes, and the explanation of different solutions for a given planning problem – to name just a few – can be implemented by advanced automated reasoning techniques.

In this article, we present the potential of hybrid planning in view of complex cognition by describing the realization of various assistance functionalities for individual users of a technical system. They include (1) generating a plan for a specific user and advising him to carry out the plan in order to achieve a current task, (2) instructing the user on how to escape from a situation where the execution of this plan unexpectedly failed, and (3) justifying and explaining the proposed solution plans in an adequate manner. These assistance functionalities can be provided by a component that relies on a domain-independent hybrid planner, a plan repair component, and a plan explanation facility.

In the scenario we have chosen to illustrate our approach, such an assistance component appears as an innovative electronic support mechanism that

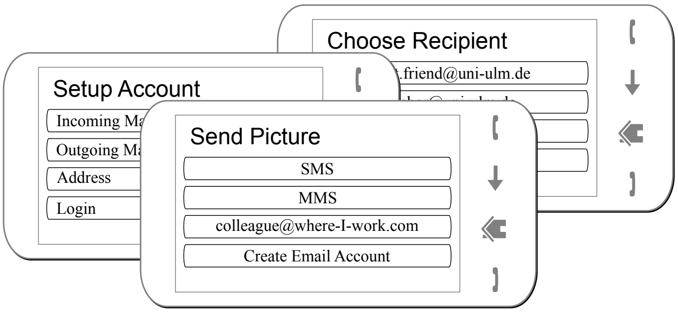


Figure 1: A schematic view of a commercially available smart phone.

smoothly helps a human user to operate his new mobile phone. In applications where the domain is a technical device, like in this scenario, setting up the planning domain model is rather straightforward. It includes the representation of relevant states and of actions that cause state transitions. Actions are applicable if certain prerequisites hold; their effects specify properties of the successor state.

The state of the phone includes aspects like the currently displayed menu and the pictures saved on the phone, but also whether it currently has reception. Our model does not account for every detail of the phone, however. Aspects below a certain level of detail are not relevant for providing support, because they are not relevant for operating the phone. For example, we do not model how the cell phone internally handles the communication with the network or signal strength; we only distinguish between having reception and not having reception.

The state of the phone can be changed by performing actions like pressing a button, activating the camera, and the like. Figure 1 shows a schematic view of the mobile phone. Performing an action like pressing the touch-screen button labeled with "MMS" changes the state of the phone. Before the button is pressed, the state of the cell phone is that it displays the send menu for pictures; afterwards, it displays the MMS dialog menu where the user can enter the details of the message to send.

As we are dealing with a deterministic technical system, all actions have a well-defined outcome. This allows us to predict future states of the mobile phone: if we know the state of the phone and the action we want to apply, we can determine its state after the application of the action. In addition, we again abstract from unnecessary details and treat action application as atomic. We thus do not need an explicit representation of time and assume that actions change the state of the phone instantaneously.

Performing a single action is usually not enough to achieve a given objective; a combination of various actions is required instead. The order in which these actions are performed generally matters. For instance, one cannot send an MMS before having entered a recipient. In some cases however, the order may not be relevant: it does not matter in which order one fills out the fields of a form, for example. Hence, the required actions are partially ordered. The particular order in which the actions have to be executed is determined by their enabling preconditions and their effects and with that imposed by causal dependencies between the actions. As a consequence, it seems to be obvious that POCL plans are an adequate means to represent the operation of a technical device.

Moreover, there are higher-level operations or tasks that comprise a whole bunch of simpler actions, an example being sending an MMS. It consists of selecting a recipient, typing in the message, and so forth. In order to perform this task, the user needs to navigate to the MMS menu, enter the recipient and various types of content, and finally has to press the send button. Those higher-level operations are not limited to a single realization; in general, there are various ways to perform them. Thus, it seems natural to use HTN representation means when dealing with these kinds of operations.

The article continues with the introduction of our formal planning framework. Section 3 presents the hybrid planning algorithm and its formal properties. Our formalizations and the respective algorithms are illustrated by a running example where a person is supported in using his new and yet unfamiliar mobile phone. In Section 4 we address the problem of unexpected environment changes and show how our plan repair mechanism is applied to help the user escaping from a situation where his intended action is doomed to failure. After that, we discuss the challenge to come up with adequate explanations of plan-based user instructions. Section 5 describes the ways in which both the causal structure of plans and the hybrid plan generation process from which they originate can be employed to achieve this objective. Finally, our presentation ends with some concluding remarks in Section 6.

2. The Hybrid Planning Framework

Hybrid planning – the fusion of partial-order causal-link (POCL) planning and hierarchical task network (HTN) planning – combines the advantages of two different approaches for solving planning problems.

POCL planning is a technique used for solving state-based planning problems. The objective is to accomplish some desired property of the world, i. e., to reach some goal state by applying actions in a correct order starting in a given initial state.

As mentioned, POCL planning is quite suitable for the purpose of finding plans that are intended to be executed by human users, since, on the one hand, the explicit information about causality helps to understand the plan and, on the other hand, the partial order of actions allows the user for more degrees of freedom in selecting the next action to execute.

HTN planning extends the principle of state-based planning to a hierarchy on the available actions. In HTN planning, actions are generally referred to as tasks. This hierarchy is established using so-called primitive tasks with preconditions and effects like in pure state-based planning, as well as abstract tasks that do not have any precondition or effect, but solely serve as containers for plans that represent predefined implementations. However, these implementations can again contain abstract tasks. The mapping between an abstract task and the plan implementing it is done by so-called decomposition methods. Thus, for each abstract task there is at least one method defining its implementation. A crucial difference between state-based and HTN planning is the solution criterion. Whereas the goal in state-based planning is to achieve a desired property, no matter which actions have to be used to accomplish this, the goal in HTN planning is to find a plan that is a valid decomposition of the initial abstract task, such that the resulting plan only contains primitive tasks.

The top-down manner in which HTN planning systems search for plans is similar to planning performed by humans when planning to achieve complex tasks like planning a business trip to a conference or setting up a complex technical device.

Most real-world application domains, like emergency evacuation and crisis management [1, Chapter 22][14] and transportation/logistics problems [16], make use of hierarchical structures on tasks and resources to a very high extent. HTN planning techniques are therefore essential to tackle those problems. Traditional HTN planning can only come up with solutions that are modeled in advance by means of decompositions for abstract tasks. In fact, HTN planning is intended to allow only these solutions, since they have been carefully designed by domain experts. We regard the procedural knowledge given in terms of decomposition methods as an enrichment of the domain model (i. e., the encoding of the world), rather than as a restriction to a subset of valid plans. Therefore, we pursue the integration of HTN planning and state-based planning, called *hybrid planning* [12, 13, 14, 15], which turned out to be well suited for solving real-world problems [17, 18].

2.1. Logical Language

Our hybrid planning formalism relies on an order-sorted, quantifier-free predicate logic. Due to space limitations we omit the definition of its *semantics* and refer to our previous work for any further details [14]. Its *syntax* is based on the *logical language* $\mathtt{L} = \langle Z, <, R, C, V, L\rangle$.

In sorted logics, all variables and constants are of some sort $z \in Z$. Table 1 lists the sorts used in our example domain model. Order-sorted logics additionally impose a hierarchy on sorts which allows for more adequate and concise formalizations. The relation $<$ is used to express this hierarchy on the sort symbols in Z. Figure 2 shows a graphical representation of a part of the sort hierarchy of our domain model. R is a Z^*-indexed family of finite disjoint sets of *relation symbols*, which are used to express properties of objects in the real world. Accordingly, C is a Z-indexed family of finite disjoint sets of *constant symbols* that represent objects in the real world. If we want to model that a specific email message is associated with a specific recipient, we use a constant like EMAIL42 of sort Email to represent this email and a constant like CONTACT23 of sort Contact to represent its recipient. The expression RecipientIsSet(EMAIL42, CONTACT23) then states the desired association. The

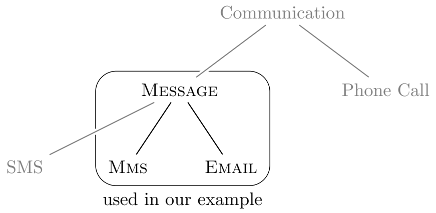


Figure 2: Part of the sort hierarchy of our smart phone domain model.

expression `Mode(USINGCAMERA)` encodes that the cell phone is in a mode in which pictures can be taken. V is a Z-indexed family of infinite disjoint sets of *variable symbols*; in our examples, variable symbols are written with a preceding question mark to distinguish them from constant symbols. To refer to the set of variables and constants of a sort $z \in Z$, we write V_z and C_z, respectively. Finally, L is an infinite set of labels used for identifying different occurrences of identical tasks.

2.2. Tasks and Plans

A *task* t is a tuple $\langle pre, post \rangle$, specifying a *precondition* and a *postcondition.* Pre- and postconditions are sets of literals over the relations of the logical language $\mathtt{L}$ and depend on the task parameters $\bar{\tau}(t) = \tau_1(t) \ldots \tau_{n(t)}(t)$, where $n(t)$ is the length of this sequence, also called the arity of task t. For convenience, we also write $t(\bar{\tau})$ to refer to a task t. For example, the task `SetUpEmail-Account(?acc1)` $= \langle \{$`Mode(SENDMENU)`, $\neg$`SetUp(?acc1)`$\}, \{$`SetUp(?acc1)`$\}\rangle$ states that setting up an email account requires the cell phone to be in the appropriate mode and that the account is not set up already; as a result, the respective email account is set up and can hence be used. We collect in $post^+(t)$ and $post^-(t)$ the atoms that occur positively and negatively in the postcondition of task t and denote them as the *positive* and *negative effects* of t, respectively.

A *state* is a finite set of ground atoms. A task t with ground pre- and postconditions is called *applicable* in a state s, if the positive literals of its precondition are contained in s and the negative ones are not. If t is applicable in a state s, its application leads to the state $s' = (s \setminus post^-(t)) \cup post^+(t)$ and is undefined, otherwise. The applicability of sequences of ground tasks is defined inductively over state sequences as usual.

Let the state of the mobile phone be $s = \{$`Mode(SENDMENU)`, `SetUp(ACC1)`$\}$. The task `SetUpEmailAccount` with its parameter `?acc1` being associated with the constant `ACC2` is applicable in s, since all positive literals of its precondition $\{$`Mode(SENDMENU)`, $\neg$`SetUp(?acc1)`$\}$ are contained in s and the negative ones are not. Its application leads to the state $s' = s \cup \{$`SetUp(ACC2)`$\}$.

A *plan* or *task network* is a tuple $P = \langle TE, \prec, VC, CL \rangle$ consisting of a finite set TE of *plan steps* or *task expressions* $te = l : t$, where t is a (partially) grounded task and $l \in L$ is a unique label to distinguish different occurrences of the same task within the same plan.

Table 1: A subset of elements of the logical language `L`, which we use to model the functionality of a smart phone.

Sorts Z

Name	Description
Mode	constants of this sort are used to represent the possible modes of the cell phone
Message	messages
Mms	MMS messages
Email	email messages
Picture	pictures
Contact	address book entries
Account	configurable email accounts

Relations R

Name	Signature	Description
`RecipientIsSet`	Message × Contact	associates a message, i. e., an MMS or email with a recipient
`Stored`	Picture	true iff a picture is stored
`Mode`	Mode	the active mode
`HasReception`		true iff the phone has reception
⋮		

Constants C

Name	Signature	Description
`USINGCAMERA`	Mode	the mode in which pictures can be taken
`SHOWALBUM`	Mode	the mode for displaying the stored pictures
`ACC1`,`ACC2`,...	Account	email accounts
`PIC1`,`PIC2`,...	Picture	pictures
⋮		

CL is a set of *causal links*, each of which has the form $\langle l_i, \phi, l_j \rangle$, indicating that the task expression te_i provides the task expression te_j with its precondition $\phi \in \mathit{post}(te_i) \cap \mathit{pre}(te_j)$, where $\mathit{post}(te)$ and $\mathit{pre}(te)$ refer to the post- and precondition of t, if $te = l : t$. This explicit representation of causal relationships between tasks is, next to our integration of hierarchical concepts, one of our main arguments for the adequacy of our formalism in our problem setting of providing assistance to human users in scenarios which involve planning capabilities because they are direct justifications for the occurrence of tasks.

Every causal link $\langle l_i, \phi, l_j \rangle$ implicitly induces an ordering between the plan steps with labels l_i and l_j. Additionally, the set of ordering constraints $\prec$ contains explicit orderings that are predefined by the model or are added as a result of the planning process. We write $\prec^*_{\mathrm{CL}}$ in infix notation when we refer to the transitive closure imposed both by the constraints from $\prec$ and those induced by *CL*. Note that $\prec^*_{\mathrm{CL}}$ defines the partial order that governs the execution of the plan – the ordering constraints $\prec$ alone are not enough.

The set of *variable constraints VC* is a set of *(non-)co-designations* used for grounding tasks and to force (in-)equality between variables. Formally, for two tasks t and t', $\tau_i(t) \doteq \tau_j(t')$ constrains $\tau_i(t)$ and $\tau_j(t')$ to be identical and, for co-designating variables with constants, $\tau_i(t) \doteq c$ constrains the variable $\tau_i(t)$ to be equal to the constant $c \in C_z$, where $z \in Z$ is the sort of c. *Non-co-designations* are defined analogously.

A *causal threat* is the situation in which the partial order of a plan would allow the plan step te_k with the postcondition $\neg\psi$ to be ordered between two plan steps te_i and te_j for which there is a causal link $\langle l_i, \phi, l_j \rangle$ such that under the current variable constraints, ϕ and ψ can be unified. This situation is a threat to the causal link, because the postcondition $\neg\phi$ would falsify the formula ϕ, thus destroying the causal link. Causal threats can be resolved by promotion or demotion (i. e., by inserting an ordering constraint $te_j \prec te_k$ or $te_k \prec te_i$, respectively), by separation (i. e., by inserting a variable constraint such that ϕ and ψ cannot unify anymore), and by expansion (because the causal threat might only exist on an abstract level; cf. Section 2.3).

Example for a Plan

Figure 3 shows a plan P_{setup} for setting up an email account in the mobile phone. It uses the following tasks:

$$\begin{aligned}
\texttt{PressEmailSetup}() &= \langle \{\texttt{Mode}(\texttt{SENDMENU})\},\\
&\qquad \{\neg\texttt{Mode}(\texttt{SENDMENU}), \texttt{Mode}(\texttt{EMAILSETUP})\}\rangle\\
\texttt{InputServerInfo}(\texttt{?acc2}) &= \langle \{\texttt{Mode}(\texttt{EMAILSETUP}), \neg\texttt{SetUp}(\texttt{?acc2})\},\\
&\qquad \{\texttt{InfoEntered}(\texttt{?acc2})\}\rangle\\
\texttt{InputCredentials}(\texttt{?acc3}) &= \langle \{\texttt{Mode}(\texttt{EMAILSETUP}), \neg\texttt{SetUp}(\texttt{?acc3})\},\\
&\qquad \{\texttt{CredentialsEntered}(\texttt{?acc3})\}\rangle
\end{aligned}$$

8

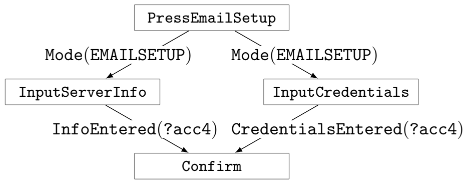


Figure 3: The plan P_{setup}, which describes how an email account can be set up in the mobile phone.

$$\begin{aligned}\texttt{Confirm(?acc4)} = \langle\{&\texttt{Mode(EMAILSETUP)}, \texttt{InfoEntered(?acc4)},\\ &\texttt{CredentialsEntered(?acc4)},\\ &\neg\texttt{SetUp(?acc4)}\},\\ \{&\neg\texttt{Mode(EMAILSETUP)}, \texttt{Mode(SENDMENU)},\\ &\texttt{SetUp(?acc4)}\}\rangle\end{aligned}$$

$P_{\text{setup}} = \langle TE_{\text{setup}}, \prec_{\text{setup}}, VC_{\text{setup}}, CL_{\text{setup}}\rangle$ is a plan where:

$$\begin{aligned}TE_{\text{setup}} &= \{l_1 : \texttt{PressEmailSetup()}, l_2 : \texttt{InputServerInfo(?acc2)},\\ &\qquad l_3 : \texttt{InputCredentials(?acc3)}, l_4 : \texttt{Confirm(?acc4)}\}\\ \prec_{\text{setup}} &= \emptyset\\ VC_{\text{setup}} &= \{\texttt{?acc2} \doteq \texttt{?acc3}, \texttt{?acc2} \doteq \texttt{?acc4}\}\\ CL_{\text{setup}} &= \{\langle l_1, \texttt{Mode(EMAILSETUP)}, l_2\rangle,\\ &\qquad \langle l_1, \texttt{Mode(EMAILSETUP)}, l_3\rangle,\\ &\qquad \langle l_2, \texttt{InfoEntered(?acc4)}, l_4\rangle,\\ &\qquad \langle l_3, \texttt{CredentialsEntered(?acc4)}, l_4\rangle\}\end{aligned}$$

Please note that the set $\prec$ of explicit ordering constraints is empty, the causal links however impose a partial order depicted in Figure 3. We include explicit ordering constraints between tasks if causal threats need to be resolved (by promotion or demotion) or if they are already present in a predefined plan of the domain model.

This plan describes which actions need to be taken in order to set up an email account. The first task is `PressEmailSetup`, which corresponds to the action of pressing the menu entry *Email Setup*. For this being possible, the mobile phone has to be in the state that actually shows the necessary menu, which is encoded by the precondition `Mode(SENDMENU)` of the task. After the execution of this action, the user has to enter the server information and the credentials, which can be done in an arbitrary order. Finally, the setup will be completed by pressing the confirmation button.

2.3. Domain Model

A *domain model* for hybrid planning is a tuple $D = \langle \mathtt{L}, \mathtt{T}, \mathtt{M} \rangle$, consisting of the logical language `L`, a set `T` of tasks and a set `M` of decomposition methods. We call a task *abstract* if there is at least one method specifying a decomposition for this task, otherwise we call it *primitive*. As an example for an abstract task, recall the task `SetUpEmailAccount(?acc1)` introduced in Section 2.2. Abstract tasks have pre- and postconditions like primitive tasks, but they do not correspond to single actions in the real world and are hence not *directly* executable by human users (they may be understandable, though; this issue will be addressed in Section 5). Instead, abstract tasks serve as containers for plans that require and achieve the pre- and postconditions of this abstract task and can thus be regarded as predefined standard solutions.

A *method* $m = \langle t, VC, P \rangle \in \mathtt{M}$ maps an abstract task t to a plan P that implements t. Each method contains a set of additional variable constraints VC to relate variables in t with variables occurring in the task network P. A method m is applied to a plan P' by replacing the abstract task t in P' by its implementation P and by inserting the variable constraints VC into the variable constraints of the plan P'. Concerning the abstract task `SetUpEmailAccount`, there is only one method that specifies how to decompose it: $m = \langle \mathtt{SetUpEmailAccount(?acc1)}, \{\mathtt{?acc1} \doteq \mathtt{?acc2}\}, P_{\text{setup}} \rangle$. This method specifies that the task `SetUpEmailAccount` can be implemented by the task network P_{setup}, that we have already seen (cf. Figure 3). The variable co-designation $\mathtt{?acc1} \doteq \mathtt{?acc2}$ ensures that the variable `?acc1` of the abstract task is properly identified with the corresponding variables used by P_{setup}. Figure 4 gives a listing of all tasks and their corresponding decomposition structure of the smart phone domain.

2.4. Problems and Solutions

A *hybrid planning problem* $\pi = \langle D, P_{\text{init}} \rangle$ consists of a *domain model* D and an *initial plan* P_{init}. The initial plan does additionally contain two artificial task expressions te_{init} and te_{goal} which are used to encode an initial and goal state, respectively. The task te_{init} has the initial state as effect and the task te_{goal} has the desired goal state as precondition. All other tasks are always ordered in between.

A plan $P_{\text{sol}} = \langle TE_{\text{sol}}, \prec_{\text{sol}}, VC_{\text{sol}}, CL_{\text{sol}} \rangle$ is a *solution* to a planning problem $\pi = \langle D, P_{\text{init}} \rangle$ if the following criteria hold:

1. P_{sol} is a refinement of P_{init}. Informally, we call a plan a refinement of P_{init}, if it results from applying modifications to it. A modification is the insertion of a plan element, i. e., an element from the set of task expressions, temporal orderings, variable constraints and causal links. The only modification that is not a pure insertion is the application of a method: it replaces an abstract task by an implementing task network and adapts the variable constraints and causal links. The formal description of the modification is introduced in the next section.
2. P_{sol} contains only primitive tasks.

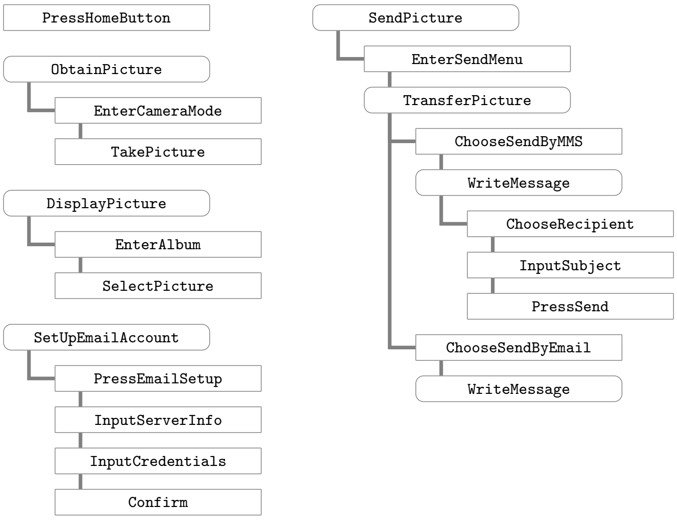


Figure 4: This figure lists the tasks inside our example domain model. It depicts the decomposition methods of abstract tasks by means of an angled line, while the subtasks of one method are connected together by straight lines.

3. All preconditions of all plan steps in P_{sol} are supported by a causal link, i. e., for each precondition ϕ of a plan step $te_j \in TE_{\text{sol}}$ there exists a causal link $\langle l_i, \phi, l_j \rangle \in CL_{\text{sol}}$ with $te_i \in TE_{\text{sol}}$.
4. There are no causal threats.
5. The ordering and variable constraints in P_{sol} are consistent, i. e., there is no plan step $te \in TE_{\text{sol}}$, such that $te \prec^*_{\text{CL}} te$ and no $v \in V_z$ for $z \in Z$, such that $VC \models v \mathrel{\dot{\neq}} v$.
6. All tasks in P_{sol} are grounded. That is, all variables are co-designated to some constant.

Solution criterion (1) is inherited from HTN planning. In this approach, any solution must be a decomposition of the initial plan P_{init}. This HTN solution criterion is reflected in hybrid planning by the requirement that solutions must be refinements of P_{init}. Since abstract tasks are regarded as non-executable, criterion (2) ensures that only executable tasks, i. e., primitive tasks, are contained in solution plans. Criterion (3) ensures the applicability of tasks in a plan: in order for a task to be applicable in a state s, all its literals of its precondition must hold in s, what can be ensured by establishing appropriate causal links. (4) guarantees that all plan steps in all linearizations of a plan are applicable in the sense of criterion (3): causal threats can cause a literal of the precondition of a plan step to be false in some linearizations although it is supported by a causal link. Since we require *all* linearizations of a plan to be valid solutions, causal threats have to be eliminated. (5) is obviously necessary for constituting meaningful plans since neither can a task be ordered before itself nor can a

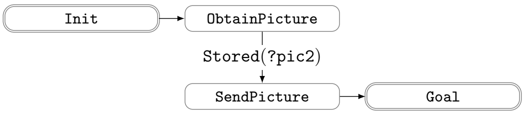


Figure 5: The initial plan P_{init} of the planning problem of our running example. It encodes James' request to take a picture and send it to a contact in his address book by the means of two abstract tasks. The initial plan contains a causal link between those tasks, because a picture has to be stored on the device, before it can be sent.

constant be different from itself. (6) maps the variables used by tasks onto the objects available in the modeled world.

Finally, we start our running example of providing plan-based support to a user. To this end, we introduce James K., the protagonist of our story.

James K. is participating in a conference on cognitive systems research. He has just attended a very interesting talk about modeling trials using ACT-R. During the session, the speaker took notes on the blackboard and James would like to discuss those with his colleagues at home. Fortunately, his new mobile phone has an integrated digital camera that seems to be up to the task. Unfortunately, the next talk is starting soon and James isn't very familiar with all those fancy functionalities, yet. Luckily, the phone is provided with an automated assistance component, on which he can rely.

This component has knowledge of the internal state of the phone and its available functions. Additionally, James can use it to query for help on sending a picture of the blackboard to one of his colleagues.

The assistance component, based on hybrid planning technology, has thus created the planning problem $\pi = \langle D, P_{\text{init}} \rangle$, with $D = \langle \mathtt{L}, \mathtt{T}, \mathtt{M} \rangle$ being the domain model and P_{init} being the initial plan, formalized as follows and illustrated in Figure 5. The initial plan P_{init} of the planning problem consists of the following tasks:

$$\begin{aligned}
\mathtt{Init}() &= \langle \emptyset, \{\mathtt{Mode(INIT)}, \mathtt{HasMobileNumber(CONTACT1)}, \\
&\qquad \mathtt{HasEmail(CONTACT1)}, \mathtt{HasReception}(), \\
&\qquad \mathtt{HasWlanConnection}()\} \rangle \\
\mathtt{ObtainPicture(?pic1)} &= \langle \emptyset, \{\mathtt{Stored(?pic1)}\} \rangle \\
\mathtt{SendPicture(?pic2)} &= \langle \{\mathtt{Stored(?pic2)}\}, \emptyset \rangle \\
\mathtt{Goal}() &= \langle \emptyset, \emptyset \rangle
\end{aligned}$$

The tasks `Init` and `Goal` are artificial tasks used to encode the beginning and the end of a plan, respectively. They thus occur only once in each plan and all other tasks are always ordered in between. The effects of the initial task `Init` encode the initial state: there is a contact (James' colleague) that is stored in the cell phone and has a mobile phone number and an email address. Also, the moment the problem gets initialized, the phone has reception and W-LAN connection. The precondition of the task `Goal` encodes the desired goal state. In our example problem, any valid decomposition of the initial plan solves the given problem without the need to explicitly satisfy a goal state.

The initial plan $P_{\text{init}} = \langle TE_{\text{init}}, \prec_{\text{init}}, VC_{\text{init}}, CL_{\text{init}} \rangle$ is formalized as follows:

$$\begin{aligned}
TE_{\text{init}} &= \{l_{\text{init}} : \texttt{Init}(), l_5 : \texttt{ObtainPicture(?pic1)},\\
&\qquad l_6 : \texttt{SendPicture(?pic2)}, l_{\text{goal}} : \texttt{Goal}()\}\\
\prec_{\text{init}} &= \{(l_{\text{init}}, l_{\text{goal}}), (l_{\text{init}}, l_5), (l_{\text{init}}, l_6), (l_5, l_{\text{goal}}), (l_6, l_{\text{goal}})\}\\
VC_{\text{init}} &= \{\texttt{?pic1} \doteq \texttt{?pic2}\}\\
CL_{\text{init}} &= \{\langle l_5, \texttt{Stored(?pic2)}, l_6\rangle\}
\end{aligned}$$

3. Plan Generation

In order to find a solution to the problem π, our hybrid planning algorithm has to refine the initial plan into a solution plan P_{sol}.

3.1. Algorithm

Our planning procedure picks a plan P from a candidate set, called *fringe*, which contains all plans that are yet to be examined. Initially, this is only the initial plan, contained in the planning problem. If P constitutes a solution the algorithm returns this plan and terminates. Otherwise, it identifies the problems with P and tries to resolve them. These problems (in the following called *flaws*) explicitly indicate, why this plan does not meet the solution criteria. For example, the second solution criterion states that all tasks of a solution plan have to be primitive. Thus, if a plan P contains abstract tasks, P raises a flaw of type *abstract task* for every abstract task in P. Obviously, the only possibility to resolve such a flaw is to select and apply an appropriate decomposition method $m \in \texttt{M}$, thus replacing the abstract task by the task network specified by m. The procedure of (1) selecting a plan, (2) identifying its flaws, and (3) applying all possible flaw-resolving modifications thereby generating a set of successor plans is repeated until a solution has been found or it has been proven that none exists. This kind of planning procedure is also referred to as refinement planning [1, 19], because each modification application specializes and hence refines the current plan.

A *flaw* is a syntactical structure that references all plan elements that are involved in the violation of a solution criterion. Then, a *flaw class* is the set of all possible flaws of a specific type and is used to relate types of flaws to appropriate types of modifications (see our previous work [20] for formal definitions). For example, the initial plan P_{init} raises (amongst others) two flaws of the flaw class $\mathcal{F}_{\texttt{AbstractTask}}$ which points to the abstract tasks `ObtainPicture` and `SendPicture`.

Table 2: This table lists the solution criteria, their corresponding flaw classes and modifications that can resolve those flaws. Solution criterion (1) is not associated with a flaw class, because any refinement algorithm automatically satisfies it.

Solution Criterion	Flaw Class	Modification Class
(1)	–	–
(2)	$\mathcal{F}_{\texttt{AbstractTask}}$	$\mathcal{M}_{\texttt{ExpandTask}}$
(3)	$\mathcal{F}_{\texttt{OpenPrecondition}}$	$\mathcal{M}_{\texttt{AddCausalLink}}$, $\mathcal{M}_{\texttt{ExpandTask}}$, $\mathcal{M}_{\texttt{InsertTask}}$
(4)	$\mathcal{F}_{\texttt{CausalThreat}}$	$\mathcal{M}_{\texttt{ExpandTask}}$, $\mathcal{M}_{\texttt{AddOrdering}}$, $\mathcal{M}_{\texttt{BindVariable}}$
(5)	$\mathcal{F}_{\texttt{InconsistentOrdering}}$	–
(6)	$\mathcal{F}_{\texttt{UnboundVariable}}$	$\mathcal{M}_{\texttt{BindVariable}}$

In order to resolve a particular flaw f that is detected in a plan P, a *modification* m is applied to P. This results in a modified or refined plan, which is free of f. A modification consists of plan elements to remove from the current plan and plan elements to insert [20]. Note that the application of m might however introduce new flaws into the plan. In our example, a modification that resolves the abstract-task-flaw `SendPicture` would remove this task together with the causal link that points to it. The task would be replaced by an implementing task network. That is, it gets replaced by a task network P for which there is a method $\langle \texttt{SendPicture(?pic2)}, \mathit{VC}, P\rangle \in \texttt{M}$. The causal link would also be replaced, since its consumer would not be the abstract task `SendPicture` anymore, but a more primitive task contained in the plan P.

A *modification class* is the set of all possible modifications of a specific type and is used to relate types of flaws to appropriate types of modifications. For instance, each modification decomposing an abstract task using its associated method belongs to the modification class $\mathcal{M}_{\texttt{ExpandTask}}$. Examples for other modification classes are $\mathcal{M}_{\texttt{InsertTask}}$ and $\mathcal{M}_{\texttt{AddCausalLink}}$, which consist of modifications inserting primitive and abstract tasks and establishing causal links between plan steps, respectively. A complete list of modification classes, as well as the flaw classes they can resolve, is given in Table 2.

As already mentioned, there is a relationship between flaw and modification classes. For example, a flaw of the class $\mathcal{F}_{\texttt{AbstractTask}}$ can only be resolved by a modification of the class $\mathcal{M}_{\texttt{ExpandTask}}$. Thus, only certain types of modifications can be used to address a specific flaw. While this allows for an efficient and distributed calculation of flaws and modifications [21], we will, for the sake of readability, only present a simplified algorithm that calculates and addresses all flaws in one step.

Our procedure is depicted in Algorithm 1; it performs search as long as there are plans in the fringe that can possibly be refined into a solution (line 1). The decision, which plan from the fringe to examine next is made by a function $\mathbf{f}^{PlanSel}$, which we call *plan selection strategy* (line 2). It implicitly defines the

Algorithm 1: Our hybrid planning search procedure.

```
Input  : The candidate set Fringe = $\{P_{\text{init}}\}$.
Output: A solution plan or fail.
while Fringe $\neq \emptyset$ do                                          1
    $P \leftarrow \mathbf{f}^{PlanSel}(\text{Fringe})$                       2
    $F \leftarrow \mathbf{f}^{FlawDet}(P)$                                   3
    if $F = \emptyset$ then return $P$                                      4
    Fringe $\leftarrow (\text{Fringe} \setminus \{P\}) \cup \{\, \mathbf{app}(m, P) \mid m \in \mathbf{f}^{ModGen}(F, P) \,\}$   5
return fail                                                                6
```

order in which the algorithm explores the space of all possible refinements of the initial plan. If the search process cannot exhaust this space completely, it depends on this function which parts will remain unexplored. $\mathbf{f}^{PlanSel}$ is also the means to introduce optimizations w.r.t. efficiency of search and quality of solution into the search process. Many different plan selection strategies have been described and evaluated in our previous work [15, 22, 20].

After P, the next plan to refine, has been chosen, all flaws are detected by the flaw detection function $\mathbf{f}^{FlawDet}$ and are stored into the set F (line 3). If P does not contain any flaws, it is considered a solution to the given problem and returned (line 4). Otherwise, the fringe is updated by removing the plan currently under consideration and by inserting its successors (line 5). To this end, the modification generation function $\mathbf{f}^{ModGen}$ computes for each detected flaw all possible modifications that resolve it. Note that there are some flaws, e. g., temporal ordering inconsistencies, that persist since they can never be resolved. Let f be such a flaw. We set $\mathbf{f}^{ModGen}(F, P) = \emptyset$ if $f \in F$ in order to discard plans containing these kinds of flaws. The calculated modifications are applied to P (by means of the function **app**) and the resulting plans are inserted into the fringe. If the fringe is empty, i. e., if there is no plan left that can possibly be refined into a solution, the algorithm leaves its main loop and returns ***fail*** (line 6). Figure 8 visualizes this procedure in the context of plan repair.

3.2. Properties of the Algorithm

In the following, we present some theoretical properties of our algorithm and sketch their proofs.

Property 1 (Correctness)**.** *Our algorithm is correct, i. e., if it returns a plan, it is a solution to the given planning problem.*

Obviously, the correctness depends on the completeness of the flaw detection and modification generation functions: every possible violation of a solution criterion must be associated with a flaw, and for every flaw all possible modifications for resolving it must be detected. If both criteria hold – which they do in our framework and implementation [15] – our claim follows by definition.

Property 2 (Completeness)**.** *If there exists a solution to a planning problem, our algorithm finds a solution, provided an appropriate plan selection strategy is chosen.*

This property is related to the fact that we perform search in an infinitely large space of plans, which may be caused by possibly recursive method applications and repeated task insertion, respectively: some tasks may be inserted, either via HTN decomposition methods or via POCL task insertion, arbitrarily often. The former is responsible for the semi-completeness of HTN planning [23], whereas the latter is a general issue of partial order planners like ours [24, 25].

Hence, a search strategy (i. e., the plan selection function $\mathbf{f}^{PlanSel}$) can "get lost" in this infinitely large search space. However, there are search strategies that guarantee to find a solution (if one exists) like the uninformed breadth-first search. Whereas informed search strategies [22, 20] do not guarantee completeness in general, in most practical scenarios they do find a solution while being much more efficient than uninformed search.

Property 3 (Termination)**.** *If no solution exists, our algorithm does not always terminate. If there is a solution, termination depends on the choice of the plan selection strategy.*

Our first claim is easy to see, since for proving the non-existence of a solution the fringe must become empty eventually. But the potential search space is infinite while only a few plans get discarded from the fringe due to unresolvable flaws; hence, termination cannot be guaranteed in cases where no solution exists. We want to emphasize that this is just a theoretical result but not of much importance for our purposes of assisting human users. This is, since we generally assume the existence of a solution and are mainly interested in finding a trustworthy, user-adapted solution quickly, rather than proving that none exists.

Our second claim follows directly from the completeness property, as the algorithm terminates as soon as a solution is encountered.

3.3. Example

Initially, the fringe contains only the initial plan P_{init}, depicted in Figure 5. The flaw detection function $\mathbf{f}^{FlawDet}$ raises three flaws: Two flaws of the class $\mathcal{F}_{\texttt{AbstractTask}}$, one for each of the two initial abstract tasks, and one flaw of the class $\mathcal{F}_{\texttt{UnboundVariable}}$, because the variable `?pic1` is not yet bound to a constant. The possible modifications detected by the modification detection function $\mathbf{f}^{ModGen}$ are applied, thereby generating a set of successor plans. At the time our planner has generated a solution, the abstract task `ObtainPicture` has been substituted by the tasks `EnterCameraMode` and `TakePicture`, and the abstract task `SendPicture` has been substituted by a plan which sends the picture via MMS. Note that in our domain model there are two methods that specify how to decompose the abstract task `SendPicture`. One method uses an MMS to send a picture, another one sends it by email. Our solution depicted in Figure 6 used the former since we assume that the assistance component makes use of a user profile in which the personal user preferences tell the system that

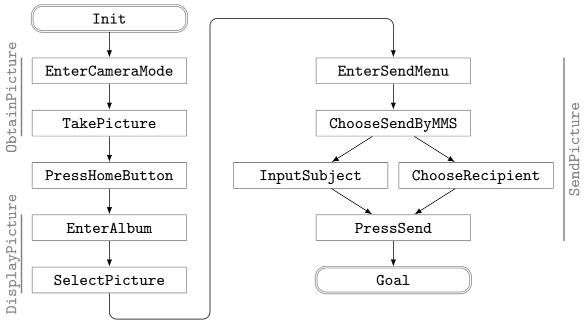


Figure 6: A solution plan to the problem of sending a picture to a contact from the address book of the smart phone. The annotations on the side describe by which abstract task the primitive tasks were introduced into the plan. The abstract tasks `ObtainPicture` and `SendPicture` were already present in the initial plan. The abstract task `DisplayPicture` and the primitive task `PressHomeButton` were introduced into the plan in order to close open preconditions of their succeeding tasks.

James has never used the email functionality before, whereas the task of sending MMS messages is quite familiar to him. Because the task `EnterSendMenu` has the precondition of the respective picture being selected, our planning system inserts the abstract task `DisplayPicture`, which in turn gets decomposed into a more primitive plan. After all flaws have been resolved, a plan has been obtained that is free of flaws and that actually is a decomposition of the initial plan.

Thus, the assistance component of James' cell phone created a solution to the problem π of taking a picture and sending it to a colleague. The solution (cf. Figure 6) is presented to James, who begins executing it.

4. Plan Repair

One basic assumption underlying our planning approach discussed so far is that the only way in which the environment changes is through actions executed by the user. Since we want to offer advanced plan-based user assistance and have to take the human user's very dynamic environment into account, this assumption is too restrictive.

James begins to follow the provided step-by-step instructions. He is able to take a picture of the notes and begins composing an MMS message to his colleague. James then leaves the room and heads for the next talk in a different room. But while doing so, the reception signal of the mobile phone gets weaker and finally completely fades.

A plan generated by the assistance component can be invalidated by exceptional events occurring at plan execution time, such as the fading reception in our example. In this section, we will present an extension to our formalism that meets the need to deal with such unforeseeable developments.

One possibility to cope with unexpected environmental changes is to restart planning from scratch, using the modified situation as the new initial state. Dealing with the problem in this way has the advantage that no modification of the planning algorithm is needed. *Plan repair*, on the other hand, tries to adapt a previously generated solution to a new situation. While plan repair is in terms of computational complexity not easier than replanning in the general case [26], the main advantage of plan repair is that it allows for more *plan stability.* Plan stability is a measure of the similarity between the original solution and the solution adapted to the new situation [27]. The goal is to present alternative plans with only minimal deviations from the original plan, as there are several reasons why plan stability is important.

First, overcoming a plan failure by generating a new solution from scratch may result in a completely different plan, which most likely appears implausible to the user and may lead to major confusion. Preserving as much of the original plan as possible instead, appears much more appropriate and helps to gain and/or preserve a user's trust. Suppose our example solution P_{sol} does not end after James has sent his message but contains further tasks, then repairing the plan will produce a plan close to the original one, while planning from scratch would only produce *some* plan.

Second, solutions to real world planning problems generally tend to be large. In case of a failure, many unexecuted parts of the plan may be unaffected. It is thus not adequate and sometimes even infeasible to create a new plan just to address a single execution failure.

Third, requests for resources or requests to third parties may have already been carried out and undoing them might lead to unfavorable effects: if something goes wrong, say, after James has already bought a flight ticket for his trip home, he is unlikely to accept a new solution that proposes to travel by train.

We will thus use a plan repair approach to deal with unpredictable environmental changes. While there exist several different plan repair approaches for classical *state-based* planning, there are hardly any for *hierarchical* planning [28]. In the following, we present our plan repair procedure for *hybrid* planning, based on our previous work [29].

Before describing the plan repair process, we will show how failures are represented in the planning domain model. In our formalism described in Section 2, the only way states can change is by the execution of actions; the formalism does not foresee external influences that cause those changes. The task of the repair procedure on the other hand is to produce a solution that compensates an unexpected failure. As a consequence, a well-founded approach to plan repair requires the means to explicitly represent failures. Unexpected changes to the environment, sometimes called *events* in the literature [1, page 5f.], can be interpreted as a kind of action themselves if we regard the environment as a second actor in the domain. These actions can of course not be used by the planner to construct plans, because they represent uncontrollable environmental developments. Instead, these so-called *processes* are inserted into the repair plan to represent execution failures. Processes invert the truth value of literals in state descriptions: the failing cell phone reception in our example is modeled

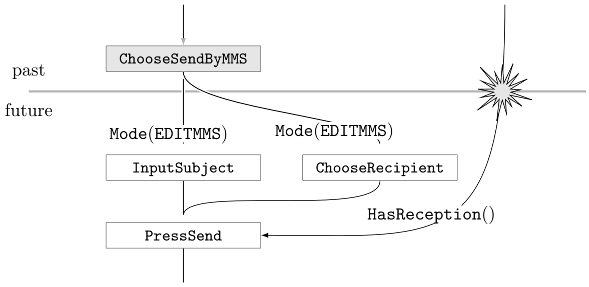


Figure 7: An example of how the execution monitor is used to monitor the evolution of the environment. The horizontal line identifies the *execution horizon*, i. e., the point in time up to which the plan has been executed. Gray nodes are tasks marked as executed by the monitor. If a failure is detected, the causal links crossing the execution horizon are candidates for being failure-affected, because their properties are needed for the execution of a future action. The causal link establishing the precondition `HasReception` for the task `PressSend` is failure-affected, as illustrated by the failure symbol on the right.

by a process $p_{\texttt{HasReception}}$ with precondition `HasReception` and postcondition ¬`HasReception`. In this way, we are able to represent all imaginable changes to the environment as a sequence of processes.

As a means to detect plan failures, we assume the existence of an *execution monitor*, which keeps track of both the execution of the actions in the plan and the evolution of the environment. When unexpected changes in the environment are discovered, it identifies which parts of the plan, if any, can no longer be executed as intended. This includes unexpected results of action execution, properties required as preconditions of future actions that do not hold as expected, and so on. The monitor maps these findings on the plan data structure and annotates the problematic parts as *failure-affected.* In our running example, the relation `HasReception` is a property required to enable a future action, namely the task `PressSend`. By the time James arrives at the room of the next talk, `HasReception` becomes false, and the execution monitor can identify the causal link establishing the precondition `HasReception` for `PressSend` as failure-affected. This situation is depicted in Figure 7. The figure also makes clear that the failure assessment is not limited to the actions immediately following the failed plan fragment; the causal structure of the plan allows us to infer and anticipate causal breakdowns for actions to be executed far in the future. We can correctly detect that the future action `PressSend` cannot be executed as intended, even if several actions (`InputSubject`, `ChooseRecipient`) lie in between.

Compared to our plan generation process presented in Algorithm 1, there is slightly more to the plan repair process, because it has to take into account that some parts of the plan are already executed and are therefore *non-retractable planning decisions.* Every executed plan element needs to occur in the repaired plan as well: James cannot go back on what he has already done. Therefore, the basic idea behind our plan repair approach is to (1) make sure that the repaired plan coincides with the failed plan on the executed parts, (2) include the exceptional environmental processes, and (3) control repair plan generation such that the new plan resembles the unexecuted parts of the original plan as well, if possible. Note that we have to include the executed parts of the original solution in the repaired plan. This is because we want it to be a solution to the original planning problem, which might include abstract tasks whose implementations are only executed in parts when the failure is detected. Thus, the solution must contain a valid decomposition of all abstract tasks included in the planning problem, cf. criterion (1) of the solution definition in Section 2.4. To make sure that executed parts of the original solution are contained unaltered in the new solution, we derive a template from the executed parts of the original solution and use it to control the generation of the repair plan. This template is represented using so-called *obligations*: for every executed plan element (like `ChooseSendByMMS` in our example), we introduce a corresponding obligation into the plan template, requiring its existence in the repair plan, including ordering and causal information. To satisfy obligations, corresponding plan elements have to be assigned to them during repair.

4.1. Problems and Solutions

Next, we will formally define plan repair problems and their corresponding solutions. For the repair problem definition $\pi^r = \langle D^r, P^r_{\text{init}} \rangle$, we first augment the domain model $D = \langle \mathtt{L}, \mathtt{T}, \mathtt{M} \rangle$ by adding the processes, resulting in $D^r = \langle \mathtt{L}, \mathtt{T}, \mathtt{M}, \mathtt{Pr} \rangle$, where `Pr` is a set of processes as follows: for every relation, we introduce two processes that invert the truth value of the respective relation from *true* to *false* and vice versa. Note that we do not simply add the processes to the set of available tasks `T`, because the system cannot actively use them to construct plans. The initial repair plan $P^r_{\text{init}} = \langle TE_{\text{init}}, \prec_{\text{init}}, VC_{\text{init}}, CL_{\text{init}}, O_{\text{init}} \rangle$ is constructed by extending the initial plan of the original planning problem $P_{\text{init}} = \langle TE_{\text{init}}, \prec_{\text{init}}, VC_{\text{init}}, CL_{\text{init}} \rangle$ with the obligations O_{init} computed from the failed solution plan P_{fail} by the execution monitor. The obligations in O_{init} have two sources. Firstly, O_{init} contains an obligation for every executed plan element (tasks, causal links, etc.) of P_{fail}. Secondly, execution failures have to be represented. Execution failures are caused by unexpected changes in the environment and may thus falsify the annotated condition of causal links. These failure-affected causal links result in appropriate obligations for processes, i. e., let $\langle l_a, \varphi, l_b \rangle$ be a failure-affected causal link, then O_{init} contains the following obligations o:

- o_a requires the existence of the task expression $te_a = l_a : t_a$,
- o_φ requires the existence of a process p_φ with precondition φ and postcondition $\neg\varphi$, encoding the environmental change,
- and finally $o_{CL} = \langle o_a, \varphi, o_\varphi \rangle$ requires the existence of a causal link between te_a and the process p_φ.

This definition is based on the assumption that all initial goals persist and that the underlying domain model is still adequate and stable. This is the case for our example, neither has anything changed that would make the domain model itself invalid nor has James changed his mind about his goals. We can thus state a plan repair problem that contains obligations up to and including the task `ChooseSendByMMS`, plus obligations as defined above for the process $p_{\texttt{HasReception}}$.

An obligation-extended plan $P^r_{\mathrm{sol}} = \langle TE_{\mathrm{sol}}, \prec_{\mathrm{sol}}, VC_{\mathrm{sol}}, CL_{\mathrm{sol}}, O_{\mathrm{sol}} \rangle$ is a solution to a plan repair problem $\pi^r = \langle D^r, P^r_{\mathrm{init}} \rangle$ if and only if the following conditions hold:

- P^r_{sol} fulfills the planning solution criteria of Section 2.4. This implies in particular that P^r_{sol} is an executable solution to the original problem, if we regard the processes added via obligations as additional tasks.

- The obligations in P^r_{sol} are *satisfied.* A set of obligations O is satisfied w.r.t. a plan P^r, if every obligation in O is assigned to a plan element in P^r such that the ordering is consistent. This ensures that the non-retractable decisions of the failed plan are respected and that the anomaly of the environment is considered.

- For all task expressions te_a and te_b, if O contains an obligation for te_b and $\mathrm{te}_a \prec^*_{\mathrm{CL}} \mathrm{te}_b$, then O also contains an obligation for te_a. This property ensures that no executed plan step can occur after an unexecuted one.

4.2. Algorithm

We need to integrate the repair mechanism into the general hybrid planning framework of Section 3 so that it can solve plan repair problems. To achieve that, we extend the flaw detection function $\mathbf{f}^{FlawDet}$ and modification generation function $\mathbf{f}^{ModGen}$ such that they detect and address unsatisfied obligations. This enables the hybrid planning Algorithm 1 to treat obligations transparently, i. e., no modification to Algorithm 1 is necessary.

Our repair procedure presented in Algorithm 2 starts at the failed original solution plan P_{fail}. This plan is extended with appropriate obligations as stated in the repair problem definition (line 1). To obtain a new solution, we make use of the previously explored plan space. This is in contrast to local search approaches that take the failed plan as-is and try to repair it by locally adding and removing tasks compensating for the failure [27, 30]. We examine the sequence of modifications *ModSeq* our planning system applied to obtain P_{fail} from the initial plan P_{init}. Plan elements are introduced by modifications, so we can determine modifications related to failure-affected plan elements. This yields a partition of *ModSeq* into two subsequences: $ModSeq_{\mathrm{fail}}$, the modifications related to failure-affected plan elements, and $ModSeq_{\mathrm{good}} = ModSeq \backslash ModSeq_{\mathrm{fail}}$ (line 2), the modifications unrelated to failure-affected plan elements. Note that sometimes, the application of a modification depends on the prior application of a modification related to failure-affected plan elements. While such

Algorithm 2: The repair algorithm for hybrid planning.

```
Input  : Repair problem $\pi^r = \langle D^r, P^r_{\text{init}} \rangle$, the failed solution plan $P_{\text{fail}}$,
         and the modification sequence $ModSeq$ leading from $P_{\text{init}}$ to $P_{\text{fail}}$.
Output: A repaired plan or fail

$P^r_{\text{fail}} \leftarrow$ addObligations($P_{\text{fail}}, \pi^r$)                                   1
$ModSeq_{\text{good}}, ModSeq_{\text{fail}} \leftarrow$ partition($ModSeq$)                          2
$P^r_{\text{retr}} \leftarrow$ retract($P^r_{\text{fail}}, ModSeq$)                                   3
$P^r_{\text{safe}} \leftarrow$ reapply($P^r_{\text{retr}}, ModSeq_{\text{good}}$)                     4
$P^r_{\text{current}} \leftarrow P^r_{\text{safe}}$                                                   5
repeat                                                                                                6
    $P^r_{\text{new}} \leftarrow$ generatePlan($P^r_{\text{current}}, \pi^r$)    // call Algorithm 1    7
    if $P^r_{\text{new}} \neq$ fail then                                                              8
        return $P^r_{\text{new}}$                                                                     9
    else                                                                                              10
        // retract last modification
        $P^r_{\text{current}} \leftarrow$ retractOne($P^r_{\text{current}}$,popLast($ModSeq_{\text{good}}$))   11
until popLast was applied to the empty sequence $ModSeq_{\text{good}}$                               12
return fail                                                                                           13
```

modifications are not directly related to failure-affected plan elements, we include them in *ModSeq*$_{\text{fail}}$. In our example, the modification that introduced the causal link establishing the precondition `HasReception` for the task `PressSend` is therefore included in *ModSeq*$_{\text{fail}}$, while the modification that added the task `InputSubject` to the plan is contained in *ModSeq*$_{\text{good}}$, because the execution of `InputSubject` does not depend on having reception. We proceed by retracting modifications in reverse order of application until we encounter P^r_{retr}, the first plan without failure-affected elements (line 3). In doing so, we have retracted all modifications in *ModSeq*$_{\text{fail}}$ and possibly some in *ModSeq*$_{\text{good}}$. The retracted modifications which are contained in *ModSeq*$_{\text{good}}$ are reapplied in line 4, yielding P^r_{safe} without involving combinatorial search. Thus, **reapply** does not have to reapply all modifications in *ModSeq*$_{\text{good}}$, but only those below P^r_{retr}. Our basic search procedure (Algorithm 1) is then started on P^r_{current}, which is initially P^r_{safe} (line 5), and attempts to refine it into a solution (line 7). If a solution P^r_{new} is found, it is returned by the algorithm. If **generatePlan** returns ***fail***, the last modification is retracted from P^r_{current} in line 11, because there might be a solution for the new initial plan P^r_{current} to which the retracted modification was not applied. If there are no more modifications to retract (line 12), the main loop terminates and the plan repair returns ***fail*** in line 11, stating that no solution exists. Figure 8 visualizes the plan space traversed during this process.

Our algorithm preserves all modifications unaffected by the failure, thereby producing a plan close to the original solution. To further improve the similarity to the original solution, a least-discrepancy heuristic can be used to guide the actual planning in line 7. These measures accomplish minimal invasiveness.

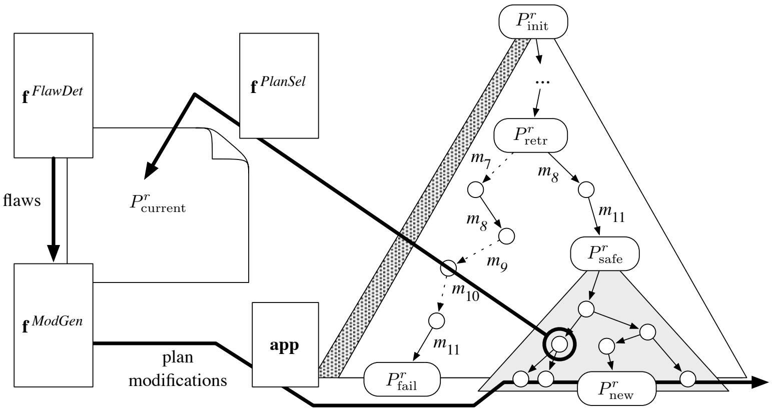


Figure 8: The left hand side sketches the hybrid planning procedure (Algorithm 1). First, a plan is selected from the fringe. For this plan, all flaws are detected, for which in turn all possible modifications are generated. These modifications are applied and the resulting plans are finally inserted into the search space depicted on the right side: It shows a visualization of the plan space traversed during the plan repair process. Modifications m_i are denoted as arrows. Dashed arrows (m_7, m_9, m_{10}) represent modifications related to failure-affected plan elements, continuous arrows (m_8, m_{11}) are unaffected modifications. Unaffected modifications are reapplied to obtain P^r_{safe}. Triangles below a plan represent the plan space reachable from that plan.

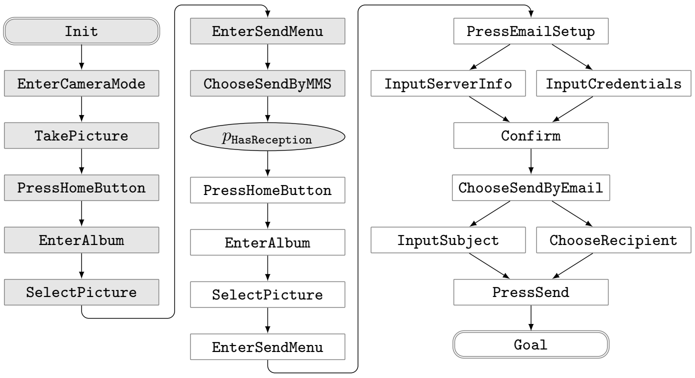


Figure 9: The result of the repair process in our example: James is instructed to set up an email account and send his picture via email using W-LAN. Nodes with a gray background denote executed tasks. $p_{\texttt{HasReception}}$ is the inserted process, representing the anomaly of the environment. Arrows denote ordering constraints. Obligations and causal links are omitted for the sake of readability.

The assistance component of the cell phone tells James that the initially generated plan cannot be carried out due to fading reception. Instead, the assistant proposes an alternative course of action: as the phone has W-LAN functionality and there is a wireless network in range (the conference offers free W-LAN access for participants), James is instructed to send his picture via email. Because he has never sent an email before from his new cell phone, he is also guided through the process of setting up an email account on the device first.

The originally generated solution (cf. Figure 6) failed due to the fading reception (cf. Figure 7). It is hence extended with obligations for the executed parts including the task `ChooseSendByMMS` and the process $p_{\texttt{HasReception}}$. The algorithm then retracts the modifications that introduced failure-affected elements. Afterwards, the resulting plan is refined to a new solution by adding tasks implementing an alternative method to send the picture. Figure 9 shows the result of the repair process in our example. It is apparent that the new solution is also a solution to the original planning problem, though it contains the process $p_{\texttt{HasReception}}$ to reflect the anomaly of the environment. Plan repair thus enables James to reach his goals despite the changing conditions.

4.3. Properties of the Algorithm

Because the main plan generation algorithm is called within the plan repair procedure (line 7), the repair algorithm's properties are basically inherited from the plan generation algorithm (cf. Section 3.2). We will thus not discuss the properties in detail but summarize them by the following statement:

Property 4 (Correctness, Completeness, and Termination)**.** *The properties Correctness, Completeness and Termination of the repair algorithm are the same as the corresponding properties of the planning algorithm as long as it terminates eventually for each problem P^r_{current} it is called with except for the root plan P^r_{init}.*

Property 3 states that the main planning procedure does not necessarily terminate in the general case, neither in cases where there is a solution, nor in cases where there is none. For our procedure, this is a problem because Algorithm 1 is called with another repair problem as soon as the previous run terminated. Pragmatically, this problem can be avoided by simply forcing termination, for instance by returning ***fail*** if no solution could be found within a predefined time limit. Termination may not be forced for the root plan P^r_{init}, as correctness and completeness would be sacrificed.

5. Plan Explanation

This section addresses the challenge of communicating an automatically generated plan to the user. This is not restricted to the way in which a solution is presented, e. g., graphically or via natural speech. More important and fundamental is the problem of getting the user to accept and understand a produced solution. We want to provide as much information as is needed for the user to trust the system and be convinced that the generated plan is appropriate to solve the given problem. Our formalism offers the hierarchical and causal structuring of planning problems and their solutions. In our opinion, this constitutes a fundamental part when aiming to reflect how humans view and think about plans, and as such is essential for conveying not only the generated plans themselves but also making the reasoning behind them transparent. While this might very well help a user to better grasp the problem and its structure, we do not focus on the didactic effect of explanation, i. e., on learning to solve the problem without technical assistance.

The issue of explaining plans that were produced by AI planning systems has not yet been addressed by the planning community. In the following paragraphs, we present our view of what some important aspects of plan explanation might be, and point out how our formalism can be used to address them. They encompass the problems of presenting plans, providing abstract views on plans, giving detailed operating instructions to perform primitive tasks, answering queries about intermediate states, justifying plan elements, and reasoning about alternative solutions.

Presenting Plans. After the planning phase is finished and a solution plan has been generated, the found solution must be presented to the user. In general there exist many ways of achieving this. We discuss the employment of the two main modalities vision and speech.

Let us go back to the moment right after James has told the assistance application that he wants to take a photo and send it to his colleague. The integrated planner has worked out a solution and now needs to present it to James. Because he will need to operate the touch screen it cannot be used for graphical output. Thus, the system uses the speaker of the phone to instruct James for his task: "Touch the camera symbol at the bottom of the screen."

Of the two modalities mentioned, the graphical option seems to be the more powerful version in general. By presenting a plan as a directed graph of plan steps that are connected by ordering constraints, it is possible to convey an impression of the complete solution at once. Figure 6 depicts the first solution of our example in such a way. By looking at a graphical presentation, one is able to assess the extent of the plan and the temporal orderings as well as possible concurrences of actions. But the expressiveness of imagery can also lead to overburdening a user with information. Especially in everyday applications, this is not a desired property.

Modalities that are less prone to the danger of providing too much information at once are text and speech. Using them to communicate a linearized version of a plan to the user results in something that resembles an ordinary manual and enables the user to execute the generated plan one step at a time. Depending on the situation, this might be the preferred mode of presentation. Additionally, the environment might prohibit the usage of a certain modality (e. g., vision when driving a car, audio in loud places).

The presentation of plans, which we consider as the most basic form of plan explanation, is also the one that has been researched the most. A recent

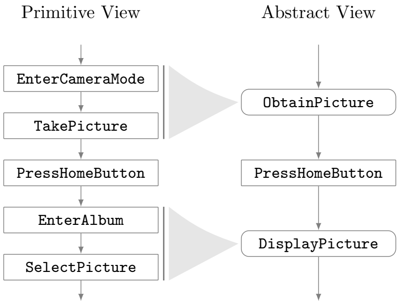


Figure 10: Primitive plan steps are summarized by the abstract tasks they were decomposed from. This results in an easier to understand representation.

publication of our group presents a working prototype system that uses a text-to-speech server and an annotated domain model to verbally present and explain plans that stem from our hybrid planning formalism [31]. A three-dimensional presentation of hierarchical plans has been investigated by Kundu et al. [32]. Another interesting approach using an ontology to describe output modalities and planning concepts to generate a visualization has been pursued by Lino et al. [33].

Operating Instructions. We are claiming that our planning formalism is well-suited to generate instructions for human users. However, the result of the planning process is a partially ordered network of plan steps, and instructions of the form `InputCredentials(ACC2)` are not easy to interpret. Additionally, the primitive tasks in a plan do not necessarily correspond to the basic actions in a one-to-one way. For example entering a subject for a message might require operating the virtual touch keyboard of the smart phone which in turn is a non-trivial series of actions. Nevertheless, the planning model abstracts from this fact and models this sequence of actions as the single primitive task `InputSubject`.

Therefore, the provision of detailed instructions for tasks is an important aspect. In order to enable non-expert users to execute the automatically generated plans, the system must be able to provide actual operating instructions. To this end, Bidot et al. [31] developed an extension of the domain model syntax of our planning formalism that enables the annotation of utterances for describing tasks via natural speech. An example in our domain could be the linking of the task `InputSubject` to the instruction “Use the virtual keyboard to enter a subject for the message into the box labeled *Subject.*”. This approach could also be extended to cover multimodal communication by providing more sophisticated markups.

Abstract Views on Plans. A plan can become rather large and conveying a comprehensive view of it is difficult. It is thus reasonable to look for ways to reduce the information contained in the presentation. One way of achieving this can be the abstraction of parts of the plan.

James is a bit annoyed because the assistance application seems to think he does not know how to take a picture. He feels very able to achieve this without help, so he demands that he is given an overview of the upcoming instructions. The screen switches to a graphical high-level presentation of the remaining plan. James can extract that he is intended to take the picture, open it again by using the album function and then transmit it by MMS. He decides to try the first two steps on his own and tells the system to turn off the instructions for a moment.

The plans generated by our planning system possess an intrinsic hierarchical structure; some plan steps have been inserted as a decomposition of an abstract task. Thus, replacing the result of a decomposition by its corresponding abstract task when presenting the plan is an obvious step. Figure 10 gives an example for an abstract view on a fragment of the solution to our example problem. The presented fragment deals with taking a picture and then displaying it in order to open the menu for sending it. Focusing on the upper half of the figure, the solution to taking a photo consists of two sub tasks: entering the camera mode of the smart phone (`EnterCameraMode`) and then triggering the camera (`TakePicture`). These two plan steps together constitute a method to implement the abstract task `ObtainPicture`. By using abstractions, the understandability of a hierarchical plan can be improved.

When reducing information in such a way, it is important to regulate the degree of abstraction applied. In more complex settings, a solution will contain deeper hierarchies of decomposition. It is an open question how a reasonable level of abstraction can be automatically determined.

When presenting abstract views on plans in an assistance application like the one in our example, information about the user should be exploited extensively. For example, an expert user might be able to execute a common task without further instructions. Thus, a task like this can be presented in an abstract way for such a user while being presented on a primitive level for novices. By using information about the level of complexity of tasks, a system can decide which abstraction to choose depending on the involved task, taking into account the general level of expertise of the user. When using a more fine-grained user model that provides an estimate of the user's abilities concerning particular tasks, it becomes possible to provide a level of abstraction that varies for each task depending on the confidence of the particular user with that task. This would be an improvement over the use of mere user classes like *expert* or *novice.*

We can summarize that the adaptation of the level of abstraction is an important tool when communicating plans, but the exact details of an ergonomic and intuitive regulation remain subject to further research.

Queries About Intermediate States. It is very important to have information about the state of the domain while executing a plan. Especially when applying planning technology to real world domains, focusing only on reaching the goal is questionable. The execution of a plan may have undesirable side-effects and it is important that those can be inspected.

Using the camera on his new phone has proven to be much easier than James had expected. The manufacturer seems to be on the right track concerning usability. However, the last step of sending the picture as an MMS remains to be undertaken. And both because he is of a very cautious nature (some people might call him paranoid) and because he wants to test the boundaries of the assistance application he asks whether the picture will still be available on the phone after having been transmitted by MMS.

In the case of totally ordered operator-based planning, obtaining the state of the domain during or after the execution of a plan can be achieved in a canonical way. Applying the plan steps one after another to the initial state, one obtains the states during the plan execution. But since we are dealing with partially ordered plans, we cannot completely determine the exact way in which the plan will be executed unless it is linearized before execution. For our example application, having the solution linearized by the system is a valid option and by doing so we can take advantage of the easier access to intermediate states. But in general, a plan produced by a hybrid planning system retains its partially ordered nature. And thus when asking questions about an intermediate or a final state of such a plan, we have to consider every linearization. This can be $n!$ linearizations for a plan consisting of n plan steps and no ordering constraints between them.

Dean and Boddy [34] have shown that the computational complexity of reasoning about states for a constellation of partially ordered events can range from polynomial time to NP-hardness; but their partially ordered events are merely conditionally executable. In contrast, all plan steps that exist in a solution to a hybrid planning problem will have their preconditions satisfied, and thus will be executable unconditionally. Therefore, we expect that the reasoning task about final and intermediate states during the execution of solutions of partially ordered plans is feasible.

The fact that we are dealing with partially ordered plans is also reflected in questions and their answers. A question like “Does property X hold after performing task T?” translates into “Does property X hold after performing task T in every linearization?”. Accordingly, answers to questions about the state of partially ordered plans reflect this as well. The response “Sometimes.” might occur very often if the question is not specific enough.

Justifying Plan Elements. Another category of questions that can be addressed are those that aim at the justification of certain plan elements. A user may be puzzled when confronted with a solution to a complex problem and may be inclined to ask why a certain plan step is necessary.

Before James was ready to hit the final send button the reception waned and the assistance application suggested a fixed solution to his problem. Since the system created the impression of a mature and well-thought-out piece of software, James simply followed the instructions guiding him along the way of *sending the picture by using the available W-LAN. At the point where he is instructed to enter the user name and password of his email account, doubts begin to emerge again and he asks why this action is a necessary step.*

Before providing an answer to such a question we have to think about what can be considered a justification. The only commitment by the user is the stipulation of the initial planning problem. We assume that this problem correctly reflects the goals that the user wants to achieve. Thus, every justification ultimately has to root there.

Taking our example, the system could respond to James' question with the immediate reason "Entering a user name and password is necessary for setting up an email account.". But the user might not be satisfied with this answer and think "Why do I have to set up an email account? I just want to send a picture to my colleague.". To close the chain of justification, the system might have better answered "Entering a user name and password is necessary for setting up an email account, which in turn is needed to send the picture by email to your colleague.".

The refinement-based hybrid planning formalism provides us exactly with this *chain of justification.* Every deviation away from the initial task network is conducted by a modification that addresses a particular flaw, and therefore can be justified by the necessity of resolving that flaw. For example, a plan step consisting of the task `InputCredentials` may have been introduced to address a flaw of type $\mathcal{F}_{\texttt{AbstractTask}}$ that was associated with a plan step `SetUpEmailAccount`. So by keeping the modification sequence that led to a solution, we can extract from it the chains of justifications that explain the necessity of certain plan elements.

To further improve on these justification chains, the system might skip certain elements inside the chain if it can be assumed that the user is able to fill the gap by using his or her own reasoning capabilities. It is also imaginable that the user model is keeping track of given explanations and this information can be used to determine which parts of a justification chain can be simplified or skipped completely. For example, if the user was already given the reason of the existence of the plan step `SetUpEmailAccount`, the chain of justification of the plan step `InputCredentials` can end there, assuming that the user has accepted the necessity of `SetUpEmailAccount`.

Reasoning about Alternative Solutions. The last paragraph has dealt with justifying the existence of plan components that constitute a found solution. This means we can substantiate the contribution of a certain element of the plan to solving the problem. This does *not* mean that this element must be present in every possible solution.

James does now understand that he has to set up his email account in order to send the picture by email. But he is not very willing to do so because he's slow with the touch keyboard. To make matters worse, the next talk has already begun and he doesn't want to spend any time on unnecessary things. Therefore, *he queries the assistance component for an alternative way to get his problem solved.*

During the plan generation process, the system makes decisions about how to resolve certain flaws that are present at an intermediate planning step. These decisions have to be supported by a user that is supposed to execute the resulting plan. A reasonable question is thus one about the reasons behind those decisions.

The hybrid planning formalism we are promoting can potentially lead to an infinite number of solutions. Depending on the domain model, there might exist a task that can be inserted arbitrarily many times without breaking the solution. For our example this might be using the home button, which resets the device to a defined state. The problem the planning system faces is to find and pick one or more of these plans for presentation to the user. If the plan space is finite and can be completely explored by the search process, answering questions about the existence of alternative solutions is trivial, giving either a positive or negative answer. Asking for a justification of why a particular solution has been preferred by the system might be more difficult and is depending on the exact way user preferences are handled. It then boils down to justifying the choice of the preference system. In the case of infinitely many solutions or a search space that cannot be completely explored because of its size, the issue becomes more and more problematic. Absolute answers are not obtainable and the system must decide how and where to spend additional computational power in order to answer questions to the best extent possible.

Of all presented aspects of the explanation of plans, the question about alternatives is the most complicated one. This is simply because it has to deal with the largest space that contributes to the answers, the complete search space. Justifying plan elements, as we are interpreting it, has to deal only with one given path through the search space – the one going from the initial problem to the solution plan. Questions about plan state are concerned only with the solution plan itself and instructions for actions only address one single plan step and maybe its parameters.

6. Conclusion

We have shown that our approach of hybrid planning enables the realization of complex cognitive capabilities of technical systems.

Automated reasoning techniques including the generation, repair, and explanation of plans can serve as core components of systems that provide advanced user assistance. In our example, such a system supports a user while he operates a mobile communication device.

User instructions are provided based on plans of action that are synthesized by the hybrid planning system. If the execution of a certain action fails due to some unexpected change of the environment, for example, the system is able to help the user out of that situation by initiating a plan repair process. The resulting plan overcomes the failure situation and is stable by exhibiting only those deviations from the original plan that are indispensable. Finally, plan explanation can be provided based on an analysis of the knowledge-rich plan structures generated by the planner as well as of the planning process itself.

Future work is devoted to developing such an explanation generator and provide it as an additional reasoning component to the envisaged assistance system. Furthermore, the question of how the system actually becomes aware of a user's intentions will be addressed. Besides the option to consult an underlying user model and compute the most likely next goal based on deposited typical user plans and the observed operation history, like it is done in plan recognition, also a speech dialogue with the user may be initiated. While in our setting, assistance is provided only in case of need when operating the device, additional modes like those where the user can explicitly ask for support or the system acts as a tutor, i. e., a proactive electronic instruction manual, would be useful enhancements of the portfolio of assistance functionalities.

Plan-based assistance is however not restricted to the support of users setting up or operating technical devices. There is a broad spectrum of prospective applications including personal organizers to support (elderly) people in their everyday decision making and medical assistance systems that accompany patients in rehabilitation processes, for example. For these kinds of applications, two additional features of our hybrid planning approach are essential and will have to be exploited: the generation of *individualized user plans* and the generation of (partially) *abstract solutions* to a given planning problem.

Acknowledgement

This work was partly accomplished within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems", funded by the German Research Foundation (DFG).

References

[1] D. S. Nau, M. Ghallab, P. Traverso, Automated Planning: Theory & Practice, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[2] R. E. Fikes, N. J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Artificial Intelligence 2 (1971) 189–208.

[3] A. L. Blum, M. L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1-2) (1997) 281–300.

[4] D. McDermott, The 1998 AI planning systems competition, AI Magazine 21 (2) (2000) 35–55.

[5] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence 129 (2001) 5–33.

[6] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (2001) 253–302.

[7] M. Helmert, The fast downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246.

[8] D. McAllester, D. Rosenblitt, Systematic nonlinear planning, in: Proceedings of the Ninth National Conference on Artificial Intelligence, 1991, pp. 634–639.

[9] J. S. Penberthy, D. S. Weld, UCPOP: A sound, complete, partial order planner for ADL, in: Proceedings of the third International Conference on Knowledge Representation and Reasoning, 1992, pp. 103–114.

[10] K. Erol, J. Hendler, D. S. Nau, UMCP: A sound and complete procedure for hierarchical task-network planning, in: Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994), 1994, pp. 249–254.

[11] Q. Yang, Intelligent Planning. A Decomposition and Abstraction Based Approach, Springer, 1998.

[12] S. Kambhampati, A. Mali, B. Srivastava, Hybrid planning for partially hierarchical domains, in: Proceedings of the 15th National Conference on Artificial Intelligence, American Association for Artificial Intelligence (AAAI Press), 1998, pp. 882–888.

[13] L. A. Castillo, J. Fernández-Olivares, A. González, A hybrid hierarchical/operator-based planning approach for the design of control programs, in: ECAI Workshop on Planning and Configuration: New results in planning, scheduling and design, 2000, pp. 1–10.

[14] S. Biundo, B. Schattenberg, From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning), in: Proceedings of the 6th European Conference on Planning (ECP 2001), Springer-Verlag, 2001, pp. 157–168.

[15] B. Schattenberg, Hybrid planning & scheduling, Ph.D. thesis, Ulm University, Germany (2009).

[16] S. Andrews, B. Kettler, K. Erol, J. A. Hendler, UM Translog: A planning domain for the development and benchmarking of planning systems, Tech. Rep. CS-TR-3487, University of Maryland (1995).

[17] T. A. Estlin, S. A. Chien, X. Wang, An argument for a hybrid HTN/operator-based approach to planning, in: Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning, 1997, pp. 182–194.

[18] L. A. Castillo, J. Fernández-Olivares, A. González, On the adequacy of hierarchical planning characteristics for real-world problem solving, in: Proceedings of VI European Conference of Planning, 2001.

[19] S. Kambhampati, Refinement planning as a unifying framework for plan synthesis, AI Magazine 18 (2) (1997) 67–98.

[20] B. Schattenberg, J. Bidot, S. Biundo, On the construction and evaluation of flexible plan-refinement strategies, in: J. Hertzberg, M. Beetz, R. Englert (Eds.), Advances in Artificial Intelligence, Proceedings of the 30th German Conference on Artificial Intelligence (KI 2007), Vol. 4667 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2007, pp. 367–381.

[21] B. Schattenberg, S. Balzer, S. Biundo, Knowledge-based middleware as an architecture for planning and scheduling systems, in: D. Long, S. F. Smith, D. Borrajo, L. McCluskey (Eds.), Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS 2006), American Association for Artificial Intelligence (AAAI Press), Ambleside, The English Lake District, UK, 2006, pp. 422–425.

[22] B. Schattenberg, A. Weigl, S. Biundo, Hybrid planning using flexible strategies, in: U. Furbach (Ed.), Advances in Artificial Intelligence, Proceedings of the 28th German Conference on Artificial Intelligence (KI 2005), Vol. 3698, Springer-Verlag, 2005, pp. 249–263.

[23] K. Erol, J. Hendler, D. S. Nau, HTN planning: Complexity and expressivity, in: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994), 1994, pp. 1123–1128.

[24] S. Kambhampati, Admissible pruning strategies based on plan minimality for plan-space planning, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 1995), 1995, pp. 1627–1633.

[25] D. E. Smith, M. A. Peot, Suspending recursion in causal-link planning, in: Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS 1996), American Association for Artificial Intelligence (AAAI Press), 1996, pp. 182–190.

[26] B. Nebel, J. Köhler, Plan reuse versus plan generation: a theoretical and empirical analysis, Artificial Intelligence 76 (1-2) (1995) 427–454.

[27] M. Fox, A. Gerevini, D. Long, I. Serina, Plan stability: Replanning versus plan repair, in: Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006), American Association for Artificial Intelligence (AAAI Press), 2006, pp. 212–221.

[28] I. Warfield, C. Hogg, S. Lee-Urban, H. Muñoz-Avila, Adaptation of hierarchical task network plans, in: Proceedings of the Twentieth Flairs International Conference (FLAIRS 2007), American Association for Artificial Intelligence (AAAI Press), 2007, pp. 429–434.

[29] J. Bidot, B. Schattenberg, S. Biundo, Plan repair in hybrid planning, in: A. Dengel, K. Berns, T. Breuel, F. Bomarius, T. R. Roth-Berghofer (Eds.), Advances in Artificial Intelligence, Proceedings of the 31st German Conference on Artificial Intelligence (KI 2008), Vol. 5243 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2008, pp. 169–176.

[30] A. Gerevini, A. Saetti, I. Serina, Planning through stochastic local search and temporal action graphs in LPG, Journal of Artificial Intelligence Research 20 (1) (2003) 239–290.

[31] J. Bidot, S. Biundo, T. Heinroth, W. Minker, F. Nothdurft, B. Schattenberg, Verbal plan explanations for hybrid planning, in: 24th MKWI related PuK-Workshop: Planung/Scheduling und Konfigurieren/Entwerfen, 2010, pp. 455–456, full article available at http://webdoc.sub.gwdg.de/univerlag/2010/mkwi/03_anwendungen/planen_scheduling/06_verbal_plan_explanations_for_hybrid_plannings.pdf.

[32] K. Kundu, C. Sessions, M. des Jardins, P. Rheingans, Three-dimensional visualization of hierarchical task network plans, in: Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space, 2002.

[33] N. Q. Lino, A. Tate, Y.-H. J. Chen-Burger, Semantic support for visualisation in collaborative AI planning, in: The International Conference on Automated Planning and Scheduling (ICAPS 2005), workshop on The Role of Ontologies in AI Planning and Scheduling, Montery, California, USA, 2005, pp. 37–43.

[34] T. Dean, M. Boddy, Reasoning about partially ordered events, Artificial Intelligence 36 (3) (1988) 375–399.