Plan and Goal Recognition as HTN Planning

Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo
Institute of Artificial Intelligence
Ulm University
D-89069 Ulm, Germany
{daniel.hoeller, gregor.behnke, pascal.bercher, susanne.biundo}@uni-ulm.de

***Abstract*—Plan- and Goal Recognition (PGR) is the task of inferring the goals and plans of an agent based on its actions. Traditional approaches in PGR are based on a plan library including pairs of plans and corresponding goals. In recent years, the field successfully exploited the performance of planning systems for PGR. The main benefits are the presence of efficient solvers and well-established, compact formalisms for behavior representation. However, the expressivity of the STRIPS planning models used so far is limited, and models in PGR are often structured in a hierarchical way. We present the approach *Plan and Goal Recognition as HTN Planning* that combines the expressive but still compact grammar-like HTN representation with the advantage of using unmodified, off-the-shelf planning systems for PGR. Our evaluation shows that – using our approach – current planning systems are able to handle large models with thousands of possible goals, that the approach results in high recognition rates, and that it works even when the environment is partially observable, i.e., if the observer might miss observations.**

***Index Terms*—Plan Recognition, Plan Recognition as Planning, HTN Planning**

I. Introduction

For systems that interact with other agents, it may be important to know which goals their counterparts want to achieve and what actions might be performed next, e.g. for planning their own behavior, or to decide when to trigger own behavior (e.g. when to offer help to a user in an assistance system). The task is commonly known as *Plan and Goal Recognition* (PGR) and it is usually solved based on a sequence of actions that the observed agent performed. There have been different approaches to solve it, e.g., rule-based, based on abduction, probabilistic inference, or techniques from parsing [1].

A more recent approach called *Plan Recognition as Planning* introduced by Ramírez and Geffner in 2009 exploits the performance of classical planning systems to solve it [2]. The main advantages are the presence of efficient solvers and well-established, compact formalisms. Since the models in PGR express the behavior of observed agents, formalisms from planning seem well-suited. However, the STRIPS models used in existing work can only express simple behavior structures similar to the regular languages known from theoretical computer science [3]. PGR is often based on hierarchical, grammar-like models (see e.g. [4]). Such models are also common in planning: the most widely used hierarchical formalism is Hierarchical Task Network (HTN) planning (see e.g. [5]). HTN models have also been proposed to be used in PGR [6], though it has been shown that the problem becomes undecidable in this setting [7]. Cardona-Rivera and Young argue that hierarchical planning formalisms bridge the gap between traditional PGR approaches and PGR as Planning [8].

HTN models combine STRIPS-like state transition with grammar-like decomposition that can express context-sensitive structures [9], combining the benefits of both. The hierarchy is often seen as means to represent *advice* in STRIPS models that represent the *physics* of a domain [10]. But it forms a second, general means of modeling (we further discuss this issue in Sec. III). The combination enables a compact yet expressive behavior representation. Sometimes it also enables a more natural model: it might e.g. be known that a certain medical treatment (that can be modeled in terms of a recipe via decomposition) leads to a good outcome, but it is difficult to model the causality behind it in terms of state transition [11]. In PGR, the hierarchy enables a natural definition of the agent's goals as the top-most tasks. These may be partially ordered, enabling the recognition of several goals of an agent with interleaving plans, or the goals of several agents.

In this paper, we present the first PGR as Planning approach in the HTN setting. It enables the recognition of complex agent behavior by using unmodified HTN planners. Our approach combines the benefits of recent work in the field like (1) recognition of *non-optimal* behavior, (2) planning only *once* instead of once for every possible goal, (3) recognizing *plans* as well as goals, (4) and the ability to deal with *missed observations*. Our evaluation shows that it works well on large plan and goal recognition corpora including thousands of goals the agent may pursue, even when the observer might miss actions.

We first summarize related work (Section II), then introduce the formal framework (Section III), before we give the formal problem definition and our approach (Section IV) that is evaluated in Section V.

II. Related Work

The first work on plan recognition as planning was introduced by Ramírez and Geffner [2] in 2009. It is based on a STRIPS model and a set of goals the observed agent may pursue. By using a (slightly modified) planning system, the approach determines those goals for which the observed actions can be part of an *optimal* solution. The observations are thereby *"... replaced by extra goals that must be achieved at no extra cost."* [2, p. 3]. The approach relies on optimal

behavior of the agent and results in a set of goals the agent may pursue. Ramírez and Geffner introduced an improved approach in 2010 with less restrictive assumptions about agent behavior [12]. The new measure is based on the *cost difference* between the two plans (with/without observed prefix). So the agent needs to approach the goal, but not necessarily in an optimal way. Now, the approach works with *unmodified* planning systems and outputs likelihoods for each goal. Like before, the planner needs to plan twice for each possible goal.

The approach of Sohrabi et al. [13] can deal with unreliable observations (e.g. missed observations) and recognizes not only goals, but also plans. Ramírez and Geffner assumed observations over actions. Sohrabi et al. also support observations over fluents. By using top-k planning and action costs, the approach does not run the planner for each goal separately.

The work by Cardona-Rivera and Young [8] is yet preliminary, but it shows that the extension to hierarchical formalisms is appealing to be able to combine recipe-like and state-based reasoning in plan recognition [8]. It introduces a PGR as Planning approach in *decompositional planning* (DP). Though it is also a hierarchical approach, it is quite different from HTN planning. In DP, the hierarchy is pure advice and solutions do not need to be included in the decomposition hierarchy. DP is equally expressive as STRIPS. Therefore they can choose the tasks to start the search from freely and integrate the observations in the starting point of their search. As a result, their overall problem class as well as approach are quite different from ours presented in this paper.

Our work introduces PGR as Planning in the setting of HTN planning and combines several advantages of related work. Like all of them, we benefit from a compact, well-established planning formalism and – using unmodified planners – do not need specialized solvers like traditional PGR approaches. Our approach does *not* rely on optimal agent behavior and needs to plan only once instead of once for every goal. It recognizes both goals *and* plans and can deal with missed observations.

III. HTN Planning

Hierarchical structures in planning are often seen as a means of introducing *advice* to non-hierarchical problems that represent *physics, not advice* [10, p. 37]. Using a hierarchy, the search space can be restricted severely, resulting in very efficient *domain-configurable* planners [14] like SHOP2 [15]. But the hierarchy adds a second, *general* means of modeling. Consider e.g. a transport domain: In a non-hierarchical model, state features describe where a transporter is located and actions change this location until it is at its desired location. The model depicted in Figure 1 shows that the same can be represented in the hierarchy: When a transporter is located at location A and shall move to C, this can be done either (1) by using a direct connection (a road between A and C) or (2) via an intermediate location B. By using these two rules (the latter being recursive), the entire part of the state representing locations can be deleted. Obviously, this hierarchy models physics, not advice. Just like state, the hierarchy can represent physics *or* advice. Thus, we consider HTN planning

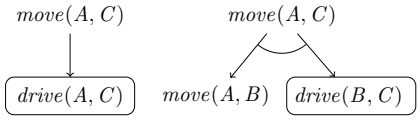


Fig. 1. The left method decomposes the abstract task $move$ into a $drive$ action that enables direct movement between two locations connected with a road. The right method is recursive and enables movement between locations without direct road connection.

a domain-*independent* approach used with *heuristic* planning systems. This is important, since it reduces the modeling effort significantly. Though such problems are harder to solve, there is a number of systems available, e.g. based on plan space [16] and progression search [17] or on compilation [18]–[20].

Plans in HTN planning must satisfy the constraints introduced by the hierarchy as well as the ones introduced by state (unlike e.g. DP). This results in a high expressivity: When interpreting the set of solutions to a planning problem as a formal language and comparing it to the Chomsky hierarchy, HTN models can express (non-context-free) context-sensitive languages, whereas STRIPS is limited to regular languages [3], [9]. So even simple bracket structures or the constraint of having one action as many times as another one can easily be represented in HTN models, but not in STRIPS.

We now introduce the HTN formalism of Geier and Bercher [21]. In HTN planning, there are two types of tasks, *compound* (or *abstract*) and *primitive* tasks (or *actions*). Abstract tasks are successively decomposed into other (abstract or primitive) tasks until only primitive tasks are left. These can be directly executed and are similar to actions in STRIPS.

The sets of primitive task names (actions) and compound task names are denoted as A and C. Tasks are organized in *task networks*. A task network tn is a triple $(T, \prec, \alpha)$. T is the (possibly empty) set of unique task *identifiers*. These are mapped to task *names* by the function $\alpha : T \to C \cup A$. This enables a single task name (e.g. *move-a-b*) to appear multiple times in a task network. The set $\prec \subseteq T \times T$ defines a partial order on the task identifiers. Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are called *isomorphic* ($tn \cong tn'$) if they differ solely in their identifiers, i.e., if there is a bijection $\sigma : T \to T'$ so that for all $t, t' \in T$ holds that $[(t, t') \in \prec] \Leftrightarrow [(\sigma(t), \sigma(t')) \in \prec']$ and $\alpha(t) = \alpha'(\sigma(t))$.

Abstract tasks are decomposed by using *(decomposition) methods*. Let M be the set of all methods. A method $m \in M$ is a pair (c, tn) that maps an abstract task $c \in C$ to a task network tn, which specifies the (primitive or abstract) *subtasks* of c and their ordering. A decomposed task is deleted from the network, the subtasks of the method are inserted and inherit its ordering relations. Formally, a method (c, tn) decomposes a task network $tn_1 = (T_1, \prec_1, \alpha_1)$ into a task network $tn_2 = (T_2, \prec_2, \alpha_2)$ if $t \in T_1$ with $\alpha_1(t) = c$ and there is a task network $tn' = (T', \prec', \alpha')$ with $tn' \cong tn$ and $T_1 \cap T' = \emptyset$.

The resulting task network tn_2 is defined as

$$\begin{aligned} tn_2 =& ((T_1 \setminus \{t\}) \cup T', \prec' \cup \prec_D, (\alpha_1 \setminus \{t \mapsto c\}) \cup \alpha') \\ \prec_D =& \{(t_1, t_2) \mid (t_1, t) \in \prec_1, t_2 \in T'\} \cup \\ & \{(t_1, t_2) \mid (t, t_2) \in \prec_1, t_1 \in T'\} \cup \\ & \{(t_1, t_2) \mid (t_1, t_2) \in \prec_1, t_1 \neq t \wedge t_2 \neq t\} \end{aligned}$$

To denote that the task network tn can be decomposed into the task network tn' by the application of zero or more decompositions, we write $tn \rightarrow^* tn'$.

The state-based part of the problem is defined over a set of propositional environment facts L. Primitive tasks hold preconditions and effects that change the current state. These are given by the functions in the triple $\delta = (prec, add, del)$, defining their preconditions, the positive and negative effects. All these functions are defined as $f : A \rightarrow 2^L$. The relation $\tau : A \times 2^L$ returns whether an action is applicable in a state with $\tau(a, s) \Leftrightarrow prec(a) \subseteq s$. When an action is applicable, the state resulting from its application is defined by the function $\gamma : A \times 2^L \rightarrow 2^L$ with $\gamma(a, s) = [s \setminus del(a)] \cup add(a)$. An action sequence $\langle a_0 a_1 \dots a_n \rangle$ is applicable to a state s_0 when each action a_i with $1 \leq i \leq n$ is applicable to the state $s_i = \gamma(a_{i-1}, s_{i-1})$.

An HTN planning domain is a tuple $D = (L, C, A, M, \delta)$. A planning problem $P = (D, s_0, tn_I)$ includes a domain D, the initial state $s_0 \in 2^L$ and an initial task network tn_I. A task network $tn_S = (T_S, \prec_S, \alpha_S)$ is a solution to a planning problem P if and only if the following conditions hold:

1) $\forall t \in T_S : \alpha_S(t) \in A$, i.e., all tasks are primitive.
2) There is a sequence $\langle t_1 t_2 \dots t_n \rangle$ of the task identifiers in T_S in line with $\prec_S$ and $\langle \alpha_S(t_1) \alpha_S(t_2) \dots \alpha_S(t_n) \rangle$ is applicable in s_0, i.e., the solution is executable.
3) $tn_I \rightarrow^* tn_S$, i.e., the solution is a refinement of the initial task network.

$Sol(P)$ denotes the set of all solutions to a problem P.

IV. Plan & Goal Recognition as HTN Planning

Based on the formal framework of HTN planning, we specify the PGR problem and introduce our approach afterwards.

A. Problem Definition

Let $D = (L, C, A, M, \delta)$ be an HTN planning domain that defines the behavior of the observed agent(s).

Having a hierarchical behavior model, goals of an agent are commonly defined as the *top-most tasks* in that hierarchy. Since an agent may pursue more than one of these tasks, we define goals in terms of a *task network*. This enables a flexible definition of goals, including multiple tasks that may or may not include the same task more than once. They may be ordered to each other. If they are not, the definition allows the plans of several goals to be interleaved. In a kitchen domain, an agent may pursue a single goal $G_p = (\{t\}, \emptyset, \{(t \mapsto makePasta)\})$ or two goals $G_{ps} = (\{t, t'\}, \emptyset, \{(t \mapsto makePasta), (t' \mapsto makeSauce)\})$. In the latter case, the subtasks of t and t' may be interleaved.

Like related work, we assume a set of possible goals, a sequence of observed actions, and the initial state to be given.

Definition 1 (PGR Problem): A *PGR problem* $(D, s_0, \bar{o}, \mathcal{G})$ extends the HTN domain model by an initial state $s_0 \in 2^L$, the observations $\bar{o}$, and a set of possible *goals* $\mathcal{G}$. Let $\bar{o} = \langle o_1, o_2, \dots, o_m \rangle$ be the sequence of observed primitive task names. We define $\mathcal{G}$ to be a (finite) set of task networks $\mathcal{G} = \{G_1, G_2, \dots, G_r\}$ where each element is defined as $G_i = (\{t_0, t_1, \dots, t_n\}, \prec, \alpha)$ with $\alpha(t_j) \in C \cup A$ and $n \in \mathbb{N}_0$.
In HTN planning, only primitive tasks are executable directly, so we assume the observations to be *primitive* tasks. Since $\bar{o}$ needs to be placed at the beginning of the solutions, we call it the *prefix* of observations. $\mathcal{G}$ contains possible combinations of top-most tasks (like G_p and G_{ps} given above).

Let $\langle a, b, c \rangle$ be the sequence of observed actions. When the environment is partially observable, a plan starting with $\langle a, y, b, z, c \rangle$ can also be considered a solution because the observer may have missed some performed actions.

Definition 2 (Goals and Plans): Given a PGR problem $(D, s_0, \bar{o}, \mathcal{G})$, some *goal* $G \in \mathcal{G}$ *explains* the observations $\langle o_1, o_2, \dots, o_m \rangle$ if and only if there is a task network $(T, \prec, \alpha) \in Sol((D, s_0, G))$ and an executable linearization $\langle t_1, t_2, \dots, t_n \rangle$ of the tasks in T with $n \geq m$ and

- for fully observable environments $o_i = \alpha(t_i)$ for $1 \leq i \leq m$, or
- for partially observable environments, there must be a strictly increasing sequence of indices $\langle j_1, \dots, j_m \rangle$ with $1 \leq j_1$, $j_m \leq n$ and $o_i = \alpha(t_{j_i})$ for $1 \leq i \leq m$.

The sequence $\langle \alpha(t_1), \alpha(t_2), \dots, \alpha(t_n) \rangle$ of primitive tasks is called the recognized *plan*.

So when an agent is executing the plan $\langle a, b, \dots, z \rangle$ and has already performed the actions $\langle a, b, c \rangle$, the sequence of observations in a fully observable environment will be $\langle a, b, c \rangle$. In a partially observable environment, actions may be missed, so the observations may be, e.g., $\langle a, c \rangle$ or $\langle b, c \rangle$.

B. Approach

This section presents our approach that, like related work, is based on a problem transformation. However, due to the HTN setting, the overall approaches differ and enforcing a prefix is more complicated. Figure 2 illustrates the overall approach. The transformation is a two-stage process, a first step introduces a new task name t_I that can be decomposed into one of the original goal-networks $G_1 \dots G_r$. Every call of the planner starts with t_I in the initial task network. For a certain goal, there may be more than one solution. The transformation makes all solutions belonging to all goals reachable. After decomposing t_I into some G_i, the system needs to find a solution by using the decompositions defined in the original model (indicated by the cloud). A second transformation guarantees that any solution includes the observations.

The planning problem that results from the overall transformation is an ordinary HTN planning problem that can be solved by any HTN planner. Solving the planning problem also solves the PGR problem: Any solution includes as first decomposition the goal G_i the planner has selected as well

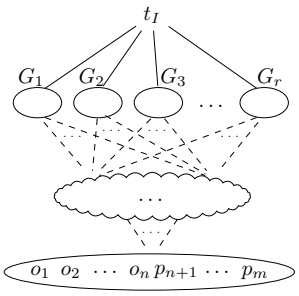


Fig. 2. Schema of the overall approach. The planner starts with a task network that contains a newly introduced task t_I that is decomposed into one of the original goal-networks G_i. Afterwards, G_i is further decomposed until a solution is found. A second transformation ensures that every solution starts with the observed prefix.

as the plan that was generated – i.e., it provides a goal that explains the observation and it recognizes the plan.

Our first transformation integrates the goal selection into the planning process and makes it possible to solve the PGR problem with a single run of the planner. The second transformation that enforces the prefix is more difficult in HTN planning. Beside modifying and extending the set of actions, we need to adapt the decomposition hierarchy. Otherwise, the new actions are not reachable for the planner. We now introduce the two transformations in more detail.

1) Selecting a Goal: Based on a given plan and goal recognition problem $PGR = (D, s_0, \bar{o}, \mathcal{G})$ with $D = (L, C, A, M, \delta)$, we define a planning problem

$$P = ((L, C \cup \{t_I\}, A, M', \delta), s_0, tn'_I)$$
$$M' = M \cup \{(t_I, tn) \mid tn \in \mathcal{G}\}$$

where $\{t_I\} \cap C = \emptyset$ is a new initial task and $tn'_I = \{\{t\}, \emptyset, \{(t \mapsto t_I)\}\}$. In the original model, $\mathcal{G}$ formed the set of possible initial task networks. Now exactly those networks can be decomposed from the newly introduced task. Therefore the following lemma holds:

Lemma 1: The set of solutions $Sol(P)$ contains *exactly* the solutions to all goals given in the original model PGR.

Next, the problem is transformed so that the set of solutions includes only those that include the given prefix.

2) Enforcing a Prefix: We build on a transformation of the *actions* similar to that known from classical planning. Then we show how to integrate the new actions into the decomposition hierarchy to make them reachable from the initial task network.

Based on a given HTN planning problem $P = ((L, C, A, M, \delta), s_0, tn_I)$ with $\delta = (prec, add, del)$, we define a problem $P' = ((L', C', A', M', \delta'), s'_0, tn'_I)$ with $\delta' = (prec', add', del')$. Let $\bar{o} = \langle o_1, o_2, \ldots, o_m \rangle$ be the sequence of observations.

To enforce the given plan prefix, we introduce new propositional symbols and duplicates of the actions with modified preconditions and effects. Let l_i with $0 \leq i \leq m$ and $l_i \notin L$ be propositional symbols that are used to place some o_i at its position in a generated plan, i.e. $L' = L \cup \{l_i \mid 0 \leq i \leq m\}$. For each task name o_i in the prefix, we introduce a new task name o'_i and define the preconditions and effects as

$$prec'(o'_i) \mapsto prec(o_i) \cup \{l_{i-1}\},$$
$$add'(o'_i) \mapsto add(o_i) \cup \{l_i\} \text{ and}$$
$$del'(o'_i) \mapsto del(o_i) \cup \{l_{i-1}\}.$$

When the environment is fully observable, every action in the original problem needs to be executed *after* the prefix, i.e., $\forall a \in A$ holds that $prec'(a) \mapsto prec(a) \cup \{l_m\}$. When the environment is partially observable, these actions remain unchanged. The new set of actions is defined as $A' = A \cup \{o'_i \mid 1 \leq i \leq m\}$. To make the first action of the prefix applicable in the initial state, the symbol l_0 is added, i.e., $s'_0 = s_0 \cup \{l_0\}$.

We want every solution to include the entire prefix. Therefore we enforce l_m to be true at the end of each plan. Due to the lack of a goal description in the HTN formalism, we introduce a new primitive task name t_G with $prec'(t_G) \mapsto \{l_m\}$, $add'(t_G) \mapsto \emptyset$ and $del'(t_G) \mapsto \emptyset$. We place this task via the initial task network after all other tasks $tn'_I = (\{t_1, t_2\}, \{(t_1, t_2)\}, \{(t_1 \mapsto t_I), (t_2 \mapsto t_G)\})$. Now, the HTN system is forced to search for plans that end with this action that holds our new goal as precondition.

So far we adapted the non-hierarchical part of the problem, but the newly introduced actions would never be reachable through decomposition, and due to the new precondition, the original actions would never be executable.

Regarding the newly introduced actions o'_i as equal to the actions they are duplicates of, it should be possible to place them at any position in the plan where o_i could have been. Therefore we introduce new abstract tasks that replace the primitive tasks in the original problem. New methods decompose these tasks either into (1) the original primitive task, or, (2) one of the new primitive tasks in the prefix.

$$C' = C \cup \{c'_a \mid a \in A\}, c'_a \notin C \cup A,$$
$$M^c = \{(c, (T, \prec, \alpha')) \mid (c, (T, \prec, \alpha)) \in M\}, \text{ where}$$
$$\forall t \in T \text{ with } \alpha(t) = n, \alpha'(t) = \begin{cases} n, & \text{if } n \in C \\ c'_n, & \text{else.} \end{cases}$$
$$M^a = \{(c'_a, (\{t\}, \emptyset, \{t \mapsto a\})) \mid \forall a \in A\},$$

Now we have to introduce a new method for every action in the sequence of observations $\bar{o} = \langle o_1, o_2, \ldots, o_m \rangle$. This method decomposes the respective new compound task c'_{o_i} into the observed action o_i:

$$M^o = \{(c'_{o_i}, (\{t\}, \emptyset, \{t \mapsto o'_i\})) \mid o_i \in \bar{o}\}$$

By defining the new set of methods $M' = M^c \cup M^a \cup M^o$, the new planning problem P' is fully specified.

Given that (1) the original primitive tasks and their corresponding duplicates are regarded equal and (2) the last artificial goal task t_G is deleted from the solution, Thm. 1 holds.

Theorem 1: Let P be an HTN planning problem, $\bar{o}$ a prefix of actions, and P' the transformed HTN problem that enforces

the prefix. Both with and without partial observability, we get that $Sol(P')$ contains *exactly* the solutions of P containing $\bar{o}$.
Proof: We first show that every solution in $Sol(P')$ contains $\bar{o}$ (in the right order). The occurrence of the tasks $o = \langle o_1, \ldots, o_m \rangle$ is enforced by the precondition l_m of the last task t_G. It can only be achieved by o_m, which requires l_{m-1} that is provided by o_{m-1}, and so forth.

In case of full observability, the original actions are intended to occur after o_m, which is enforced by its newly introduced precondition l_m.

We need to show that all solutions in $Sol(P)$ containing $\bar{o}$ are also in $Sol(P')$. The set of primitive task networks that are reachable does not change (when considering an observed action and its duplicate equal): We replaced the actions in P by new compound tasks, but also introduced two methods for it; one decomposes the task into the original action and the other into its duplicate encoding the observation.

If a (primitive) solution task network of P contains the observed (original) actions in the correct order, then clearly there is an isomorphic solution in P', where these actions are substituted by their duplicate pendent. The duplicates perfectly mimic the original effects, but additionally ensure that they are executed in the right order. For full observability, the encoding ensures that all original actions are executed after the prefix. □

3) Overall Transformation: Given a PGR problem $(D, s_0, \bar{o}, \mathcal{G})$, we compile D, s_0, and $\mathcal{G}$ into a planning problem P by using the first transformation and enforce $\bar{o}$ by the second transformation, resulting in P'. Taking Lemma 1 and Theorem 1 together, we get:

Corollary 1: Let PGR be a plan and goal recognition problem and P' an HTN planning problem resulting from applying the transformations. Then, $Sol(P')$ contains *exactly* the solutions given in Definition 2.

Instead of constructing a specialized plan and goal recognition system to solve the task, we can create the corresponding HTN problem and pass it on to any HTN planning system.

In HTN planning, the planner is not allowed to insert actions apart from the hierarchy, it has to choose a goal with a plan that includes an observed action in the first place. Among plans in line with the observations, the planner will tend to choose a cheap plan according to its cost measure. The agent is assumed to act according to the HTN model, but not necessarily optimal. The hierarchy is often regarded to specify combinations of actions that are *helpful strategies* to reach a goal, i.e., that are *rational behavior*. So the assumptions about goal-directed behavior are given in the domain.

Like related work, we rely on full information about the initial state and deterministic actions. We use our transformation to generate a *single* solution. Our evaluation shows that the strategy of finding a goal with a short plan that includes the observations in the first place works quite well in practice. However, we consider it the natural next step to generate a distribution of probabilities, e.g. by using top-k planning like Sohrabi et al. [13].

V. Empirical Evaluation

a) Domains: There are only few evaluation corpora for PGR available. Beside the actual model for recognition, pairs of goals and plans that belong to each other are needed and the domain should incorporate a set of qualitatively different goals. As first evaluation domain we have chosen the Monroe domain[1]. It is a disaster management domain and was built by Blaylock and Allen [22] to generate a corpus of 5000 plan/goal combinations. They generated the plans by using a modified SHOP2 planning system that was adapted to randomly generate different possible plans for a given goal. It includes 10 different lifted goals, 46 lifted methods and 30 lifted actions and a huge set of constants that result in nearly 2000 distinct groundings of the goals in the generated corpus. For details, we refer to their paper given above. From a computational complexity view, the domain is neither totally ordered, nor (lifted/grounded) acyclic, nor tail-recursive, i.e., properties that would make it decidable [5], [23], are not fulfilled. We use an HTN domain that is widely based on their SHOP2 domain.

To contribute to the portfolio of available benchmark corpora, we created a new corpus[2]. We wanted to model a problem with the following characteristics:

1) It should intrinsically incorporate multiple goals that are pursued by the agent, i.e. that contribute to a single plan,
2) it should include partial ordering between tasks, and
3) it should be difficult to recognize because plans for several goals share a common prefix.

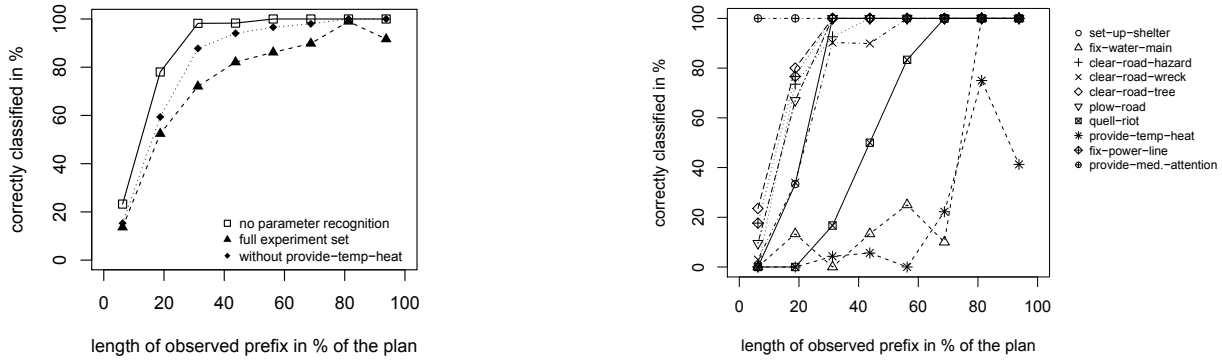
We created a Kitchen domain including 5 distinct starters, 30 distinct main dishes, and distinct 5 desserts. The main dishes itself are combinations of different food (i.e. a main dish implies several goals), e.g., making noodles and a pasta sauce. This means that even two main dishes might share a common prefix of actions. The overall goal is to either create • a main dish, • a starter and a main dish, • a main dish and a dessert, or • a starter, a main dish and a dessert.

This leads to more than 1000 different goal combinations in the corpus (without counting just grounding matters like which cooking pan is used for which food). By construction, many of them share the same prefix and many steps are partially ordered. Consider e.g. seeing a cook making a certain starter and a certain main dish, then the plan recognizer can never be sure if it has observed the full plan (cooking these two things) or the prefix of making a meal that additionally contains a dessert. The mean number of goals included in each instance is 4.27. The used domain intrinsically incorporates tasks that are partially ordered. We analyzed the overall domain and found a combination of starter, main dish, and dessert that results in more than 2×10^{37} distinct linearizations of the solution.

When using the same planning system to generate the corpus and for plan recognition, there might be a bias in the results. Since we use a search-based planner to solve the *recognition* task, we used our SAT-based planner [19] to

[1] https://www.cs.rochester.edu/research/speech/monroe-plan/

[2] Our software is available online at www.uni-ulm.de/en/in/ki/panda.



(a) Over all goals.

(b) Split by (ground-truth) goal.

Fig. 3. Goal recognition in the Monroe domain (fully observable setting). Curves represent the percentage of correctly classified goals given a certain percentage of the prefix has been seen. Each point represents the ratio between instances with correctly recognized goal and instances with wrongly recognized goal in an interval of 12.5% of the prefix length. Consider, e.g., the right-most triangle in the left plot at position (93.75, 86.4). It means that, from instances with an enforced prefix of a length between 87.5% and 100% of the ground-truth plan, 86.4% of the goals are classified correctly.

generate the corpus. It is a satisficing planner, i.e. the plans may be sub-optimal. The generated plans include a mean value of 35.4 actions.

b) Experiment Setup: We implemented a lifted version of the transformation[2] and used 100 instances of each corpus. In the Monroe domain, the plans of these instances have a length of 4 to 29 actions (mean value 10.7); in the Kitchen domain a length of 16 to 50 (mean value 36.7). For each instance, we generated one planning problem with enforced prefix of length 0, 1, 2, and so on, until the whole original plan has been enforced. The planner was started with the transformed problem and a single solution was generated. From this result, the goal and the generated plan was extracted as the *recognized* goal and plan.

In principle, the planning system that is used is of minor importance for our approach. We used the most recent version of our heuristic search system PANDA [17] to solve the resulting PGR problems. It ran on Xeon E5-2660 v3 CPUs with 2.60 GHz base frequency, 16 GB memory, and a time limit of 10 minutes.

A. Fully Observable Environments

In the Monroe domain, the planner solved 84% of the instances. Due to the large set of constants, the median of the peak memory for grounding was 2.12 GB, but only 0.17 GB for search. The median runtime was 43 seconds. In the Kitchen domain, the planner solved 99% of the instances. The median of the peak memory (grounding and search) was 0.47 GB. The median runtime was 3.7 seconds. I.e., from a planning perspective, the Kitchen domain is easier to solve. This is due to the large set of constants in the Monroe domain that made grounding costly.

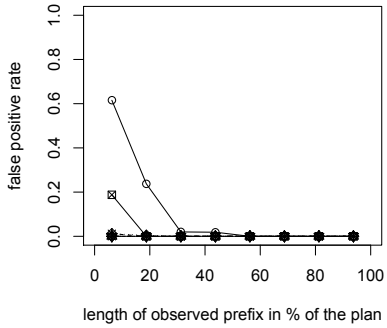


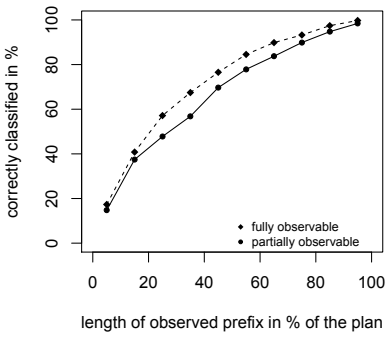
Fig. 4. False positive rate, split by type of goal, ignoring parameters (labels as given in Figure 3b).

1) Goal Recognition: Figure 3a shows the performance of the goal recognition over all solved instances and all goals in the Monroe domain. On the x-axis it shows the length of the observed prefix, on the y-axis the percentage of instances with correctly classified goal. Due to the different plan lengths, the length of the prefix is given as *percentage* of the overall plan.

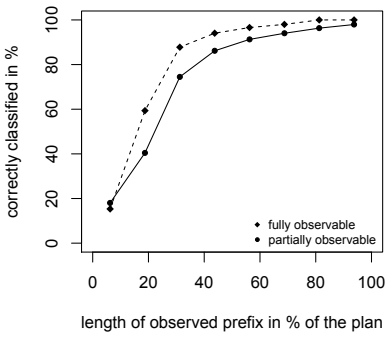
The bottom curve shows the recognition rate on the full tasks, i.e., the result is correct when the task and all parameters have been recognized correctly. This results in thousands of possible classifications (nearly 2000 of them are included in the corpus). To further investigate the cause of incorrect classifications, we analyzed whether the *task* was classified correctly (i.e., incorrect parameters were ignored, this results in 10 possible classifications). The percentage of correctly classified tasks is given in top-most curve. Then, the goal is recognized correctly after a short prefix. We come back to the third curve later.

Studying the first plots, we were wondering why the system could detect the tasks, but not the correct parameters in so many cases. So we had a look at the performance on distinct goals. Naturally, there are goals that result in sequences of actions that are more unique than others, revealing the goal. Other goals may result in plans that do not really distinguish one goal from another. So one might assume that some goals are recognized after a short prefix, others might need a longer prefix to be recognized.

We split the data on the full recognition (task and parameters) by the different goal tasks. The results are given in Figure 3b. Some of the goals are recognized quite well after observing a short prefix, others only when the half or



(a) Kitchen domain.



(b) Monroe domain.

Fig. 5. Comparison of the goal recognition results in the fully and partially observable setting.

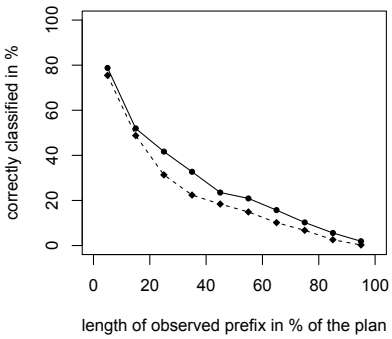


Fig. 6. False positive rate.

even three quarters of the overall plan have been observed. Quite notable is the poor recognition performance on the goal *provide-temp-heat*. We identified the following method in the domain used for generation that does cause it:

```
(:method m-provide-temp-heat-1
  :parameters (?person - person
               ?ploc - point)
  :task (provide-temp-heat ?person)
  :precondition (and
    (atloc ?person ?ploc))
  :ordered-subtasks (and
    (generate-temp-electricity ?ploc)
    (turn-on-heat ?ploc)))
```

It decomposes the task *(provide-temp-heat ?person)* into two subtasks and includes a method precondition[3] that binds the parameter *?ploc* to the location of the person to treat. The subtasks ensure that this location is provided with electricity and heat, but the person itself does not appear in the plan. Given that more than one person may be at this location, it is impossible for a PGR system to infer the right one. Surely, such structures should be avoided when designing a model for PGR (though it is totally fine for planning, the purpose the domain was designed for by Blaylock and Allen).

Now we can come back to Figure 3a. The curve in the middle gives the performance over all goals except for *provide-temp-heat*. Now the overall performance of recognizing the goal with all parameters nearly reaches the performance of recognizing only the goal task without parameters.

We want to have a look at the *false positive rate*. Obviously, this can not be provided for each single *grounding* of the goals. Instead, Figure 4 shows it separate for each goal task, ignoring all parameters. In the first interval (including the case with no observed prefix), the planning system tends to choose *set-up-shelter* as goal. This is in line with Figure 3b, where the result for this task is very high from the start. So the plans for this goal might be easy to find for the planner.

Figure 5a and 6 show the goal recognition results for the Kitchen domain. The dotted curves show the recognition results for the fully observable setting that is discussed here (be will come back to the other curves later on). Figure 5a shows the percentage of ground-truth goals that have been recognized correctly. It can be seen that the recognition in this domain is more difficult than the Monroe domain. In this domain, the planner has to choose the number of goals, e.g. if there is solely a main dish, or an extra dessert. This can result in false positives. Figure 6 gives the false positives, i.e., the percentage of goals that have been returned by the recognition system but are not included in the set of ground-truth goals. The dotted curve gives, again, the totally observable setting. It can be seen that the false positive rates decrease quite fast when the number of observations increases.

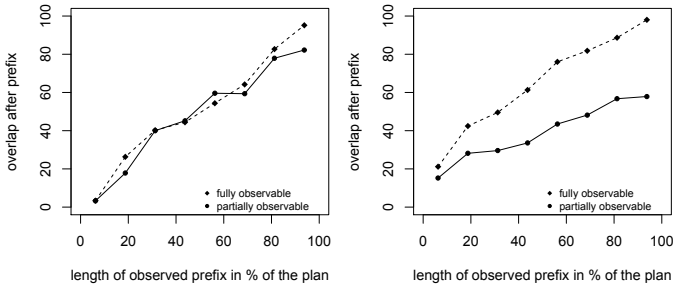
2) Plan Recognition: Plan recognition provides further information about the observed agent's behavior. The ground-truth plan and the recognized plan have necessarily the enforced prefix of observations in common. Figure 7 and 8 give the percentage of actions after the enforced prefix (denoted the postfix) that are identical (over all runs, regardless on whether the goal was recognized correctly or not). For both domains, it gives (1) the percentage of the remaining ground-truth actions that are included in the recognized plan and (2) the percent of the remaining actions in the recognized plan are included in the ground-truth plan. It can be seen that the results for the second value are better – we think that this is caused by the planner's tendency to find *short* plans. Though the evaluated domains enable a large set of possibilities on how to achieve a goal, the system finds plans with a large percentage of actions included both in the generated and in the ground-truth plan.

B. Partially Observable Environments

An interesting question is if our approach can deal with partially observable environments, causing the observer to miss actions. Therefore we performed a second series of experiments. We generated partially observable experiments by randomly deleting 20% of the observations. In the Monroe domain, we used the set excluding *provide-temp-heat* and 15 minutes time limit. However, the median solving time increased only slightly to 49 seconds for Monroe and it stays under 4 seconds in the Kitchen domain. The coverage in the Monroe domain decreased to 61%, in the Kitchen domain to 85%.

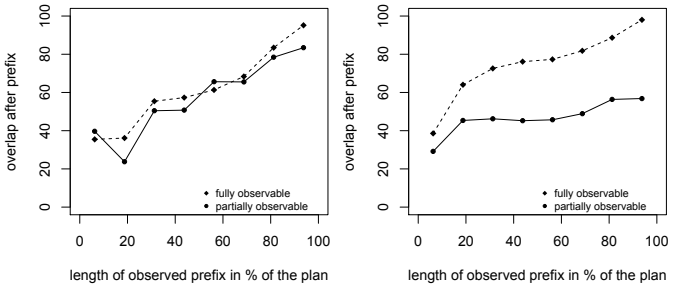
1) Goal Recognition: Figure 5b shows a comparison between the recognition rate in the fully and partially observable

[3]*Method preconditions* are a feature of the planner SHOP that is commonly compiled away before planning.



(a) Monroe domain. (b) Kitchen domain.

Fig. 7. Percentage of the postfix ground-truth actions that are included in the recognized plan.



(a) Monroe domain. (b) Kitchen domain.

Fig. 8. Percentage of the postfix actions of the recognized plan that are included in the ground-truth plan.

environment for the Monroe domain. It is lower in the beginning, but becomes nearly as accurate as in the fully observable setting when the observed prefix becomes longer. A similar result can be seen in the Kitchen domain (Figure 5a).

2) Plan Recognition: Figure 7 and 8 include the plan recognition results for the partially observable setting (the non-dotted lines). Most notably is the drop of the plan recognition results in the Kitchen domain. We think that this is caused by the higher false positive rate causing more non-ground-truth actions in the generated plan.

Overall, it can be seen that the approach can handle missed observations, especially in goal recognition.

VI. Conclusion

Plan and Goal Recognition as Planning exploits efficient solvers and well-established, compact formalisms for behavior representation from planning in the field of plan and goal recognition. We are the first who introduced the approach in the field of HTN planning. The use of HTN instead of STRIPS models enables the recognition of far more complex behavior (context-sensitive instead of regular structures), thereby combining widely-used grammar-like models with state-transition as given in STRIPS. Our approach combines several advantages from recent work in the field: (1) Recognition of non-optimal behavior, (2) planning only *once* instead of once for every possible goal, (3) recognizing *plans* as well as goals, (4) and the ability to deal with *missed observations*. Our evaluation shows that (1) our approach works well on corpora containing a large set of possible goals; that (2) unmodified, off-the-shelf HTN planners can solve the resulting PGR problems and that (3) the approach results in good recognition rates, even in partially observable environments, i.e. when the observer might not see all executed actions.

References

[1] G. Sukthankar, R. P. Goldman, C. Geib, D. V. Pynadath, and H. H. Bui, *An Introduction to Plan, Activity, and Intent Recognition*. Elsevier, 2014.

[2] M. Ramírez and H. Geffner, "Plan recognition as planning," in *Proc. of IJCAI*. IJCAI/AAAI Press, 2009, pp. 1778–1783.

[3] D. Höller, G. Behnke, P. Bercher, and S. Biundo, "Assessing the expressivity of planning formalisms through the comparison to formal languages," in *Proc. of ICAPS*. AAAI Press, 2016, pp. 158–165.

[4] C. W. Geib and R. P. Goldman, "Recognizing plans with loops represented in a lexicalized grammar," in *Proc. of AAAI*. AAAI Press, 2011.

[5] K. Erol, J. A. Hendler, and D. S. Nau, "Complexity results for HTN planning," *Annals of Mathematics and Artificial Intelligence*, vol. 18, no. 1, pp. 69–93, 1996.

[6] C. W. Geib, "Assessing the complexity of plan recognition," in *Proc. of AAAI*. AAAI Press/The MIT Press, 2004, pp. 507–512.

[7] G. Behnke, D. Höller, and S. Biundo, "On the complexity of HTN plan verification and its implications for plan recognition," in *Proc. of ICAPS*, 2015, pp. 25–33.

[8] R. E. Cardona-Rivera and R. M. Young, "Toward combining domain theory and recipes in plan recognition," in *Proc. of the AAAI Workshop on Plan, Activity, and Intent Recognition*, 2017.

[9] D. Höller, G. Behnke, P. Bercher, and S. Biundo, "Language classification of hierarchical planning problems," in *Proc. of ECAI*. IOS Press, 2014, pp. 447–452.

[10] D. V. McDermott, "The 1998 AI planning systems competition," *AI Magazine*, vol. 21, no. 2, pp. 35–55, 2000.

[11] R. Goldman, "A semantics for HTN methods," in *Proc. of ICAPS*. AAAI Press, 2009, pp. 146–153.

[12] M. Ramírez and H. Geffner, "Probabilistic plan recognition using off-the-shelf classical planners," in *Proc. of AAAI*. AAAI Press, 2010.

[13] S. Sohrabi, A. V. Riabov, and O. Udrea, "Plan recognition as planning revisited," in *Proc. of IJCAI*. IJCAI/AAAI Press, 2016, pp. 3258–3264.

[14] D. S. Nau, "Current trends in automated planning," *AI Magazine*, vol. 28, no. 4, pp. 43–58, 2007.

[15] D. S. Nau, T. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.

[16] P. Bercher, G. Behnke, D. Höller, and S. Biundo, "An admissible HTN planning heuristic," in *Proc. of IJCAI*. ijcai.org, 2017, pp. 480–488.

[17] D. Höller, P. Bercher, G. Behnke, and S. Biundo, "A generic method to guide HTN progression search with classical heuristics," in *Proc. of ICAPS*. AAAI Press, 2018, pp. 114–122.

[18] A. D. Mali and S. Kambhampati, "Encoding HTN planning in propositional logic," in *Proc. of the 4th Int. Conf. on Artificial Intelligence Planning Systems*. AAAI Press, 1998, pp. 190–198.

[19] G. Behnke, D. Höller, and S. Biundo, "Tracking branches in trees – A propositional encoding for solving partially-ordered HTN planning problems," in *Proc. of ICTAI*. IEEE Computer Society, 2018.

[20] R. Alford, G. Behnke, D. Höller, P. Bercher, S. Biundo, and D. W. Aha, "Bound to plan: Exploiting classical heuristics via automatic translations of tail-recursive HTN problems," in *Proc. of ICAPS*, 2016, pp. 20–28.

[21] T. Geier and P. Bercher, "On the decidability of HTN planning with task insertion," in *Proc. of IJCAI*. AAAI Press, 2011, pp. 1955–1961.

[22] N. Blaylock and J. F. Allen, "Generating artificial corpora for plan recognition," in *Proc. of the 10th Int. Conf. on User Modeling (UM)*. Springer, 2005, pp. 179–188.

[23] R. Alford, P. Bercher, and D. W. Aha, "Tight bounds for HTN planning," in *Proc. of ICAPS*. AAAI Press, 2015, pp. 7–15.