New Perspectives on Cost Partitioning for Optimal Classical Planning

Inauguraldissertation
zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

FLORIAN POMMERENING

aus Tübingen, Deutschland

Basel, 2017

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Malte Helmert,
Universität Basel, Dissertationsleiter, Fakultätsverantwortlicher

Prof. Dr. J. Christopher Beck
University of Toronto, Korreferent

Basel, den 23.05.2017

Prof. Dr. Martin Spiess,
Universität Basel, Dekan

Abstract

Admissible heuristics are the main ingredient when solving classical planning tasks optimally with heuristic search. There are many such heuristics, and each has its own strengths and weaknesses. As higher admissible heuristic values are more accurate, the maximum over several admissible heuristics dominates each individual one. *Operator cost partitioning* is a well-known technique to combine admissible heuristics in a way that dominates their maximum and remains admissible. But are there better options to combine the heuristics? We make three main contributions towards this question:

Extensions to the cost partitioning framework can produce higher estimates from the same set of heuristics. Cost partitioning traditionally uses non-negative cost functions. We prove that this restriction is not necessary, and that allowing negative values as well makes the framework more powerful: the resulting heuristic values can be exponentially higher, and unsolvability can be detected even if all component heuristics have a finite value. We also generalize operator cost partitioning to transition cost partitioning, which can differentiate between different contexts in which an operator is used.

Operator-counting heuristics reason about the number of times each operator is used in a plan. Many existing heuristics can be expressed in this framework, which gives new theoretical insight into their relationship. Different operator-counting heuristics can be easily combined within the framework in a way that dominates their maximum.

Potential heuristics compute a heuristic value as a weighted sum over state features and are a fast alternative to operator-counting heuristics. Admissible and consistent potential heuristics for certain feature sets can be described in a compact way which means that the best heuristic from this class can be extracted in polynomial time.

Both operator-counting and potential heuristics are closely related to cost partitioning. They offer a new look on cost-partitioned heuristics and already sparked research beyond their use as classical planning heuristics.

Zusammenfassung

Zulässige Heuristiken sind ein Hauptbestandteil für das optimale Lösen von klassischen Handlungsplanungsproblemen mit heuristischer Suche. Es gibt viele solche Heuristiken und jede hat ihre eigenen Stärken und Schwächen. Da höhere zulässige Heuristikwerte genauer sind, dominiert das Maximum über mehrere zulässige Heuristiken jede einzelne. *Operatorkostenpartitionierung* ist eine bekannte Technik um zulässige Heuristiken auf eine Art zu kombinieren, die deren Maximum dominiert und zulässig bleibt. Aber gibt es noch bessere Optionen um die Heuristiken zu kombinieren? Wir machen drei wichtige Beiträge zu dieser Frage:

Erweiterungen der Kostenpartitionierung erlauben es die gleichen zugrundeliegenden Heuristiken zu höheren zulässigen Werten zu kombinieren. Kostenpartitionierung verwendet traditionellerweise nicht-negative Kostenfunktionen. Wir beweisen, dass diese Einschränkung nicht nötig ist und dass das Framework mächtiger wird, wenn auch negative Werte zugelassen werden: die resultierenden Heuristikwerte können exponentiell grösser werden und Unlösbarkeit kann auch erkannt werden, wenn alle Teilheuristiken endlich sind. Wir verallgemeinern Operatorkostenpartitionierung ausserdem zu Transitionskostenpartitionierung was es erlaubt zwischen verschiedenen Situationen zu unterscheiden, in denen ein Operator angewandt wird.

Operator-counting-Heuristiken werden aus Aussagen darüber berechnet, wie oft ein Operator in einem Plan verwendet wird. Viele existierende Heuristiken können in diesem Framework ausgedrückt werden, was uns interessante Einblicke in ihre Beziehungen erlaubt. Verschiedene Operator-counting-Heuristiken können innerhalb des Frameworks einfach miteinander kombiniert werden, so dass die Kombination deren Maximum dominiert.

Potentialheuristiken berechnen ihren Heuristikwert als gewichtete Summe über Zustandseigenschaften und sind eine schnelle Alternative zu Operator-counting-Heuristiken. Zulässige und konsistente Potentialheuristiken für bestimmte Mengen von Zustandseigenschaften können auf eine kompakte Art beschrieben werden, die es erlaubt die beste Potentialheuristik in polynomieller Zeit zu extrahieren.

Operator-counting- und Potentialheuristiken sind eng mit Kostenpartitionierung verwandt. Sie eröffnen neue Perspektiven auf kostenpartitionierte Heuristiken und haben schon jetzt Forschung in Anwendungen jenseits von klassischen Planungsheuristiken angeregt.

Acknowledgements

First and foremost, I would like to thank Malte Helmert. I cannot imagine having a better PhD advisor. I learned so much from him during my time in Basel and continue to be amazed by how quickly he completely understands something after listening to just a few rambling sentences of mine. He always took the time to help even when his own workload was crushing.

I would also like to thank Chris Beck for agreeing to join my thesis committee as my second reviewer despite very tight time constraints from my side and a time consuming exam period.

I am also very grateful to all my colleagues in Basel: Salomé Eriksson, Cedric Geissmann, Manuel Heusner, Thomas Keller, Gabi Röger, Jendrik Seipp, Silvan Sievers, and Martin Wehrle. They provided a fantastic working environment and many long and interesting discussions. I am specifically grateful for their support during the last months, which gave me the breathing room to finish this thesis.

I owe a great debt to Tom Mayer, Sven Wehner, Cedric Geissmann, Thomas Keller, and Salomé Eriksson for proof reading, finding my mistakes, and helping me improve this text, all on short notice.

Last but not least, I would like to thank my family – Bodo Pommerening, Karla Oechelhaeuser, and Thilo Pommerening – and my friends (most of all Ingo Stößer, Marlène Birk, and Tom Mayer) for their continuous encouragement and support.

Thank you!

Contents

1. Introduction

The aim of automated planning (Ghallab, Nau, and Traverso, 2004) is to find a sequence of actions that achieves a given goal. In a logistic problem, for example, actions could be to drive trucks from one place to another, load packages into trucks, and unload them again. A solution would then be a sequence of load, unload, and drive actions that ensure that every package is delivered to its target destination. Examples for planning tasks span a wide variety of applications: deciding which elevator to send to which floor, scheduling activities for satellites and Mars rovers, solving puzzles like the sliding tile puzzle, playing combinatorial games like Sokoban or FreeCell, and many more.

Research on domain-independent planning investigates techniques that solve planning problems without making assumptions about the specifics of the problem domain. A domain-independent planner reads the description of the available actions, the initial state of the world and the desired goal conditions, and returns a sequence of actions achieving the goal from the initial state; no matter if the actions model loading a package into a truck, analyzing a rock sample with a rover, or moving a card in a solitaire game.

Here, we mostly focus on *classical* planning, where actions have discrete, deterministic and fully observable effects, and the plan is created offline for a single agent who executes actions in sequence. In particular, we consider *optimal classical planning*, where the solutions must come with a guarantee that no cheaper solution exists.

Optimal domain-independent planning is often solved with A* search (Hart, Nilsson, and Raphael, 1968). A* search iteratively chooses an unexplored state (starting with the initial state of the world) and *expands* it: for every action that is applicable in the state, one *successor* is generated that represents the state reached by applying the action. The choice of which state to look at next is guided by a *heuristic function* (Pearl, 1984) that estimates the cost of reaching a goal state from a given state. The sum of the heuristic value of a state and the cost to reach the state is an estimate for the cost of a plan via this state called the f-value. A* search always expands states with minimal f-value. If the heuristic never overestimates the cheapest cost to reach a goal state, then A* is guaranteed to find an optimal solution. Such a heuristic is called *admissible*. Even though there are some exceptions (Holte, 2010), searching with admissible heuristics that produce higher values generally requires expanding fewer nodes.

There are many admissible heuristics for optimal classical planning. Although some heuristics are provably at least as accurate as others, it often depends on the domain which heuristic performs best. The choice between several heuristic can be avoided by using all of them. For example, most state-of-the-art solvers use multiple abstractions

to cover different problem aspects. However, this brings up a new question which is central to this thesis:

How can several admissible heuristics be combined
in a way that guarantees admissibility?

There are two established ways to answer this question. The maximum of a set of admissible heuristics is always admissible and at least as good as any one of the heuristics. Operator cost partitioning (Katz and Domshlak, 2010b) is an alternative that combines the heuristics by evaluating each one on a copy of the task with a suitably reduced cost function. Optimal cost partitioning achieves values that are at least as good as the maximum and often much higher.

This thesis investigates extensions to operator cost partitioning that produce higher admissible estimates from the same component heuristics. We also introduce two new heuristic families that offer a new perspective on cost partitioning, compute higher quality heuristic combinations more quickly, and give new insight on existing heuristics.

1.1. Contributions

We make three main contributions which are all related to cost partitioning and heuristic combination.

Operator cost partitioning has traditionally used only non-negative costs. Our first main contribution is to prove that allowing negative costs as well maintains admissibility and can dramatically improve heuristic quality. Cost partitioning with general cost functions can extract information from heuristics that report a value of 0; it can produce exponentially high heuristic values even if each component heuristics only reports values that are polynomial in the size of the task; and it can detect that a task is unsolvable even if all heuristics have a finite value. All of this is not possible with non-negative cost functions. We also extend operator cost partitioning in an orthogonal way by partitioning the costs of individual transitions instead of whole operators. This allows us to treat operators differently depending on the context in which they are used.

The second main contribution is the family of *operator-counting heuristics*. These heuristics unify many existing heuristics in a common framework. Operator-counting heuristics can easily by combined with each other in a way that dominates their maximum. We prove that this combination is a form of cost partitioning. This connection helps us to better understand the involved heuristics. By analyzing their constraint structure, we can show interesting connections between them and explain or improve their performance. The framework of operator-counting heuristics is very general, so it can be applied in many other areas as well. This way insights from classical planning can be transferred to other planning formalisms.

Our third main contribution are *potential heuristics*, another new family of heuristics which offer a fast alternative to operator-counting heuristics. Potential heuristics have a

fixed mathematical structure and desired properties can be expressed as constraints on this structure. We show how to completely characterize interesting sets of heuristics in a way that the best heuristics from these sets can be efficiently extracted. The simple structure of potential heuristics makes them useful in other areas as well: we show how they can be used to detect whether a task is solvable or to categorize how difficult a task is. Potential heuristics are also closely related to cost partitioning and thus provide yet another way of combining admissible heuristics.

1.2. Structure

Before we start with the main topic of this thesis, Chapter 2 formally introduces classical planning and heuristic search. The family of *abstraction heuristics* plays an important role in most of the thesis and is also introduced in Chapter 2. We discuss other heuristics as they become relevant. Throughout the thesis, we use linear and mixed integer programs. Chapter 3 gives a brief overview of the topic and introduces basic terms and notation. A deeper understanding of linear programs is certainly useful, but not required to read the rest of this thesis.

The main body of this thesis consists of three parts about *cost partitioning*, *operator-counting heuristics*, and *potential heuristics*. The structure of all three parts is mostly parallel and consists of the following chapters:

- Introduction of the basic concept
- Extensions to the basic concept
- (Only in Parts II and III) Deeper theoretical analysis of the heuristics introduced in the previous two chapters. Here, we also connect each part to the previous parts and show how operator-counting heuristics and potential heuristics relate to cost partitioning.
- Experimental evaluation
- (Only in Parts II and III) Discussion of application areas outside of heuristic search for optimal planning where the heuristics are already used or could be used in the future.
- Summary

Chapter 20 finally concludes the thesis and discusses how the three parts fit together.

1.3. Experimental Setup

Each part of this thesis concludes with an experimental evaluation of the described techniques. To this end, we implemented the techniques in the Fast Downward planning system (Helmert, 2006). All of our experiments share the same basic setup.

We use all tasks from the optimal tracks of the international planning competitions (IPC) 1998–2014 as a benchmark set. This set consists of 1667 tasks from 57 domains. The set contains duplicates because some domains and tasks were reused in different IPCs. We did not remove such duplicates as the full set is an established benchmark set in the planning literature. Tasks are given as PDDL files (Fox and Long, 2003). We use the translator that is part of Fast Downward to translate them to SAS$^+$ tasks (see Section 2.1) in a preprocessing step.

We use the experiment framework downward-lab (Seipp et al., 2017) for all experiments. Unless otherwise indicated, we limit runtime to 30 minutes and memory usage to 2 GB. All experiments ran on a cluster of machines with Intel Xeon E5-2660 processors (2.2 GHz) and each task was limited to a single core. The number of concurrently running tasks on different cores of the same CPU can differ over time. When an experiment starts, all cores are used but towards the end of an experiments some cores idle. This can influence the runtime of a task slightly (usually below 5%). To avoid systematic bias, we therefore randomized the order in which tasks were started.

Linear programs and mixed integer programs were solved with CPLEX v12.5.1 (IBM, 2017). We access the CPLEX library through the open solver interface (OSI v0.107.8) of the COIN-OR project (Lougee-Heimer, 2003).

We use the number of tasks solved within the resource bounds as a primary measure of performance and call this the *coverage* of a configuration. In addition, we often evaluate runtime and heuristic quality. The runtime excludes the time taken by Fast Downward's preprocessor as this step is identical for all configurations we consider.

Heuristic quality can be measured as the heuristic value of the initial state or as the number of expansions of an A* search. When we report expansions, we always exclude the expansions of the last f-layer because their number is influenced heavily by tie-breaking. We often report initial heuristic values in scatter plots where each mark in the plot represents one task and its coordinates are the heuristic values achieved by different heuristics. We restrict such plots to values up to 50 because in the domain ParcPrinter operator costs are so high that initial heuristic values are in the order of 10^5–10^6. Showing such high values in the same plot would hide the detail for a majority of tasks. Values larger than 50 are projected down to 50.

1.4. Relation to Published Work

Most of the results reported in this thesis are published in conference proceedings of major conferences on automated planning and artificial intelligence. I now briefly sum-

marize the most important publications and how they relate to my thesis.

- Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014b. LP-based heuristics for cost-optimal planning. In *Proc. ICAPS 2014*, 226–234.

  We introduced operator-counting heuristics as a framework that unifies existing heuristics based on linear programming. These heuristics form the core of Part II and contributions from the paper occur in Chapters 8, 9, and 11, as well as in Section 10.3.

  The paper won the **ICAPS 2014 outstanding paper award**.

- Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proc. AAAI 2015*, 3335–3341.

  We generalized the cost partitioning and connected it to operator-counting heuristics. Cost partitioning is the main topic of Part I and relevant contributions from the paper occur in Chapters 4, 5, and 6. I discuss the connection to operator-counting heuristics in Chapter 10. The paper also introduces potential heuristics, which are the topic of Part III and contribute to Chapter 14, Sections 15.1 and 17.1. The accompanying technical report (Pommerening et al., 2014a) proves that a simplifying syntactical restriction can be lifted, which I do in Appendix B.

  The paper won the **outstanding paper award for AAAI 2015**.

Apart from these two papers that form the core of my thesis, my contributions to the following papers are also used here.

- Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In *Proc. IJCAI 2013*, 2357–2364.

  We proposed *post-hoc optimization*, a method to combine heuristic estimates from abstraction heuristics, which we later integrated into the operator-counting framework. Post-hoc optimization is discussed in Section 9.3.

- Pommerening, F., and Helmert, M. 2015. A normal form for classical planning tasks. In *Proc. ICAPS 2015*, 188–192.

  We argued that many ideas in automated planning are easier to describe and understand if the tasks are assumed to be in a normal form and showed that general tasks can be transformed into this form with small linear overhead. This normal form is introduced in Section 2.4 and used throughout the thesis to simplify presentation.

- Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In *Proc. ICAPS 2015*, 193–201.

  We optimized potential heuristics for different objectives like a high initial or a high average heuristic value. This is covered in Sections 15.4 and 17.1.

- Keller, T.; Pommerening, F.; Seipp, J.; Geißer, F.; and Mattmüller, R. 2016. State-dependent cost partitionings for Cartesian abstractions in classical planning. In *Proc. IJCAI 2016*, 3161–3169.

  Among other contributions we defined transition cost partitioning, which is introduced in Sections 5.2 and 5.3 here.

- Pommerening, F.; Helmert, M.; and Bonet, B. 2017b. Higher-dimensional potential heuristics for optimal classical planning. In *Proc. AAAI 2017*, 3636–3643.

  We investigated potential heuristics for different sets of features, after only considering the set of atoms in previous work. This contributed to Sections 15.2, 15.3, and 17.2. We also found a connection between potential heuristics and transition cost partitioning, the last piece of the puzzle of connecting the three parts of this thesis, which we discuss in Section 16.2. The paper also makes a contribution in the area of constraint optimization that is mostly unrelated to planning but required for a proof. Here, an extended proof is in Appendix C.

- Pommerening, F.; Helmert, M.; and Bonet, B. 2017. Abstraction heuristics, cost partitioning and network flows. In *Proc. ICAPS 2017*. Accepted for publication.

  We investigated network flow heuristics and how to compactly represent them. Here, we discuss this contribution in Sections 9.6 and 10.4.

In addition to the papers above, I also discuss my contribution to the following publications that are more tangential to the main topic of my thesis.

- Heusner, M.; Wehrle, M.; Pommerening, F.; and Helmert, M. 2014. Under-approximation refinement for classical planning. In *Proc. ICAPS 2014*, 365–369.

  We under-approximated a task by considering only a subset of operators. We suggested to use operator-counting techniques for selecting the operators that should be included in the under-approximation and mention this in Section 12.1.

- Röger, G.; Pommerening, F.; and Helmert, M. 2014. Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In *Proc. ECAI 2014*, 765–770.

  We suggested context splitting, which plays a role in defining transition cost partitioning in Section 5.2.

- Röger, G., and Pommerening, F. 2015. Linear programming for heuristics in optimal planning. In *AAAI 2015 Workshop on Planning, Search, and Optimization*, 69–76.

  We summarized current LP-based heuristics and already described the example task from Section 2.1 and the LP formulation from Section 9.2 there.

- Seipp, J.; Pommerening, F.; Röger, G.; and Helmert, M. 2016a. Correlation complexity of classical planning domains. In *Proc. IJCAI 2016*, 3242–3250.

  We used potential functions to define a measure of complexity for planning tasks. Here, we describe it in Section 18.1.

- Seipp, J.; Pommerening, F.; Sievers, S.; Wehrle, M.; Fawcett, C.; and Alkhazraji, Y. 2016b. Fast Downward Aidos. In *Unsolvability International Planning Competition: planner abstracts*, 28–38.

  Our submission to the first unsolvability planning competition (UIPC 2016) was a portfolio called *Aidos*. Aidos includes heuristics based on potential functions, which we discuss in Section 18.2.

  Aidos was the **winner of the UIPC 2016**, outperforming 10 competitors.

2. Classical Planning and Heuristic Search

Heuristic search with admissible heuristics is one of the most common approaches to find optimal solutions for classical planning tasks. This chapter formally introduces classical planning tasks and planning heuristics. We also introduce the family of *abstraction heuristics*, which is used throughout the thesis, and a normal form for planning tasks, which we use to simplify presentation later on.

2.1. Planning Tasks

Throughout this thesis, we use the simple logistics task shown in Figure 2.1 as an example for various techniques. In this task a truck that is initially at location A has to pick up a package that is initially located at location B and bring it back to A. At the end, the truck should be parked at location B. While this task is trivial, it is sufficiently complex as an example in most cases.

We consider planning tasks, like this example, that can be encoded in the SAS$^+$ formalism (Bäckström and Nebel, 1995). SAS$^+$ tasks are based on a set of finite-domain *variables* $\mathcal{V}$, where each $V \in \mathcal{V}$ is associated with a finite set of values $\mathit{dom}(V)$ called its *domain*. The example task can be encoded using two variables: *pos-T* with values $\mathit{dom}(\textit{pos-T}) = \{A, B\}$ to encode the position of the truck, and *pos-P* with values $\mathit{dom}(\textit{pos-P}) = \{A, B, T\}$ to encode the position of the package. A tuple like $\langle \textit{pos-P}, B\rangle$ encodes the fact that the package is at location B. Such tuples are called *atoms*.

Definition 2.1 (atom)**.** *Let $\mathcal{V}$ be a set of finite-domain variables. An* atom *is a tuple $\langle V, v\rangle$ of some variable $V \in \mathcal{V}$ and one of its values $v \in \mathit{dom}(V)$. The set of all atoms over $\mathcal{V}$ is $\mathcal{A} = \{\langle V, v\rangle \mid V \in \mathcal{V}, v \in \mathit{dom}(V)\}$.*

Situations and conditions in our modeled world can be described by mapping the

Figure 2.1: Example logistics task: the truck can drive between A and B, and the package can be loaded in and unloaded from the truck.

variables to values in their domain. For example, the state shown in Figure 2.1 could be described as $\{\textit{pos-T} \mapsto A, \textit{pos-P} \mapsto B\}$.

Definition 2.2 (variable assignments and states)**.** *Let $\mathcal{V}$ be a set of finite-domain variables. A partial function $f : \mathcal{V} \to \bigcup_{V \in \mathcal{V}} \textit{dom}(V)$ is called a* variable assignment *over $\mathcal{V}$ if $f(V) \in \textit{dom}(V)$ for all variables V in the domain of f. We write the domain of a variable assignment f as $\textit{vars}(f)$.*

A variable assignment over $\mathcal{V}$ is also called a partial state *and a partial state s with $\textit{vars}(s) = \mathcal{V}$ is called a* state*. The set of all states over $\mathcal{V}$ is $\mathcal{S}$.*

Depending on what is clearer in each context, we interpret a variable assignment s either as a (partial) function, e.g. $\{\textit{pos-T} \mapsto A, \textit{pos-P} \mapsto B\}$, or as a set of atoms, e.g. $\{\langle \textit{pos-T}, A\rangle, \langle \textit{pos-P}, B\rangle\}$. When we interpret s as a function, we write $s[V]$ instead of $s(V)$ to avoid confusion with other parentheses. We say an atom $\langle V, v\rangle$ *is true in state* s if $s[V] = v$. To fully model our example, we also need to describe what actions an agent can perform.

Definition 2.3 (operator)**.** *Let $\mathcal{V}$ be a set of finite-domain variables. An* operator *over $\mathcal{V}$ is a tuple $o = \langle p, e\rangle$ where both the* precondition $pre(o) = p$ *and the* effect $\textit{eff}(o) = e$ *are partial variable assignments over $\mathcal{V}$.*

In our example task there is an operator *drive-A-B* to drive the truck from A to B with the precondition $\{\textit{pos-T} \mapsto A\}$ and the effect $\{\textit{pos-T} \mapsto B\}$ and an analogous operator *drive-B-A* to drive back. Additionally, there is an operator *load-B* to load the package into the truck at location B with precondition $\{\textit{pos-T} \mapsto B, \textit{pos-P} \mapsto B\}$ and effect $\{\textit{pos-P} \mapsto T\}$ and analogous operators *load-A*, *unload-A*, and *unload-B*.

Where $\mathcal{V}$ is clear from context, we usually just talk about atoms, (partial) states, and operators and leave "over $\mathcal{V}$" implied. We can now formally define SAS$^+$ planning tasks.

Definition 2.4 (planning task)**.** *A* planning task *is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, \textit{cost}\rangle$ with the following components:*

- *$\mathcal{V}$ is a finite set of finite-domain* variables*,*
- *$\mathcal{O}$ is a set of* operators*,*
- *s_{I} is a state, called the* initial state*,*
- *$s_\star$ is a partial state, called the* goal condition*, and*
- *$\textit{cost} : \mathcal{O} \to \mathbb{N}_0$ is the* cost function*.*

The initial state of our example task is $s_{\mathrm{I}} = \{\textit{pos-T} \mapsto A, \textit{pos-P} \mapsto B\}$ and the goal is $s_\star = \{\textit{pos-T} \mapsto B, \textit{pos-P} \mapsto A\}$. The operators are *drive-A-B*, *drive-B-A*, *load-A*, *load-B*, *unload-A*, *unload-B* and all have a cost of 1.

We now turn to the semantics of a planning task. We say a state s is *consistent* with a partial state p if $p \subseteq s$, i.e. if $s[V] = p[V]$ for all $V \in \textit{vars}(p)$.

Definition 2.5 (operator application)**.** *An operator o is* applicable *in a state s if s is consistent with $pre(o)$. If o is applicable in s then the result of* applying *o in s is the state $s[\![o]\!]$ with*

$$s[\![o]\!][V] = \begin{cases} \mathit{eff}(o)[V] & \text{if } V \in \mathit{vars}(\mathit{eff}(o)) \\ s[V] & \text{otherwise.} \end{cases}$$

A sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ is applicable in a state s if there are states $s_0, \ldots, s_n$ with $s = s_0$, o_i is applicable in s_{i-1} and $s_{i-1}[\![o]\!] = s_i$ for all $1 \leq i \leq n$. The result of this application is the state $s[\![\pi]\!] = s_n$.

The only applicable operator in the initial state of our example task is *drive-A-B* and the state $s_{\mathrm{I}}[\![\textit{drive-A-B}]\!]$ is $\{\textit{pos-T} \mapsto B, \textit{pos-P} \mapsto B\}$. The operator sequence $\pi = \langle \textit{drive-A-B}, \textit{load-B}, \textit{drive-B-A} \rangle$ is also applicable in s_{I} and ends in the state $s_{\mathrm{I}}[\![\pi]\!] = \{\textit{pos-T} \mapsto A, \textit{pos-P} \mapsto T\}$. The cost of applying an operator o is $\mathit{cost}(o)$ and the cost of applying an operator sequence $\pi = \langle o_1, \ldots, o_n \rangle$ is $\mathit{cost}(\pi) = \sum_{1 \leq i \leq n} \mathit{cost}(o_i)$.

States that are consistent with the task's goal condition are called *goal states* and we are looking for operator sequences that reach a goal state:

Definition 2.6 (plan)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, \mathit{cost} \rangle$ be a planning task and s be one of its states. An* s-plan *is an operator sequence $\pi = \langle o_1, \ldots, o_n \rangle$ that is applicable in s and ends in a state that is consistent with $s_\star$. An s_{I}-plan is just called a* plan*.*

An optimal s-plan under cost function *cost′ is a s-plan π where $\mathit{cost}'(\pi)$ is minimal among all s-plans. An* optimal s-plan *is an s-plan that is optimal under the cost function of Π.*

The only optimal plan in the running example task is $\langle$*drive-A-B*, *load-B*, *drive-B-A*, *unload-A*, *drive-A-B*$\rangle$ with a cost of 5. Under an alternative cost function where loading and unloading is free of cost (e.g. if we are interested in minimizing fuel consumption), this still is an optimal plan but has a cost of 3 and is no longer the only optimal plan.

A task Π induces a weighted, labeled *transition system* $\mathrm{TS}_\Pi = \langle \mathcal{S}, \mathcal{T}, s_{\mathrm{I}}, \mathrm{S}_G \rangle$ with the set of states $\mathcal{S}$, the initial state s_{I}, the set of goal states S_G that contain all states consistent with $s_\star$, and the following set of transitions $\mathcal{T}$: for each $s \in \mathcal{S}$ and each $o \in \mathcal{O}$ that is applicable in s, there is a transition $s \xrightarrow{o} s[\![o]\!] \in \mathcal{T}$ labeled with o and weighted with $\mathit{cost}(o)$. The sequence of labels from transitions along a shortest path in TS_Π correspond to an optimal plan for Π. We define a state $s \in \mathcal{S}$ as *alive* in TS if there is a path from s_{I} to s and from s to a goal state in S_G. Otherwise, we call the state *dead*. The transition system of our example task is shown in Figure 2.2.

Optimal planning is the problem of finding an optimal plan for a planning task Π or proving that no plan for Π exists. One commonly used approach to optimal planning is *heuristic search*.

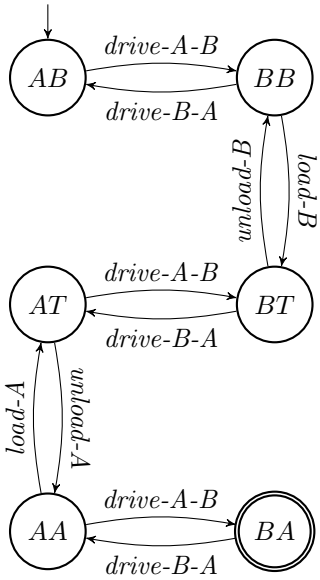


Figure 2.2: Transition system of our example task. A node XY represents the state $\{\textit{pos-T} \mapsto X, \textit{pos-P} \mapsto Y\}$. The initial state is marked with an incoming arrow, the goal state is marked with a double line.

2.2. Heuristic Search

In this section, we introduce the notion of a heuristic, and related concepts that are relevant to this thesis. In the following section, we then introduce a family of heuristics called abstraction heuristics, which are used throughout the rest of the thesis.

A *heuristic* estimates the cost of an optimal s-plan for some state s of a planning task Π. Heuristics are often defined as functions $h : \mathcal{S} \to \mathbb{N}_0 \cup \{\infty\}$, but here we consider the term in a more general way. We allow heuristics to take any value from $\mathbb{R} \cup \{\infty, -\infty\}$ and also consider functions that depend on more parameters. We consider functions that depend on a state and a cost function with the intuition that $h(s, \textit{cost}')$ estimates the cost of an optimal s-plan under cost function *cost*′. We omit *cost*′ if it is the cost function of the task, i.e. $h(s) = h(s, \textit{cost})$.

The *optimal heuristic* h^* maps each state $s \in \mathcal{S}$ and cost function *cost*′ to the cost of an optimal s-plan under *cost*′ or to ∞ if no such plan exists.

Definition 2.7 (heuristic properties)**.** *A value $c \in \mathbb{R} \cup \{\infty, -\infty\}$ is an* admissible heuristic estimate *for state s and cost function cost′ if $c \leq h^*(s, \textit{cost}')$. A heuristic h is* admissible *if $h(s, \textit{cost}')$ is an admissible heuristic estimate for all states s and cost functions cost′.*

A heuristic function h is goal-aware *if $h(s_G, \textit{cost}') \leq 0$ for every goal state s_G and every cost function cost′.*

A heuristic function h is consistent *if $h(s, cost') \leq cost'(o) + h(s[\![o]\!], cost')$ for every state s, operator o applicable in s and cost function $cost'$.*

Heuristics that are goal-aware and consistent are admissible (Russell and Norvig, 1995), and admissible heuristics are goal-aware. A* search (Hart, Nilsson, and Raphael, 1968) with an admissible heuristic generates optimal plans. Admissible heuristics with higher heuristic estimates produce estimates that are closer to the optimal heuristic. They generally lead to less search effort, although this not guaranteed (Holte, 2010). We say a heuristic h_1 *dominates* another heuristic h_2 if $h_1 \geq h_2$. We say that the dominance is strict, if the inequality is strict for at least one state. If h_1 is admissible and dominates h_2, then obviously h_2 is also admissible.

If multiple admissible heuristics are available, A* can always use their maximum:

Proposition 2.1. *Let $h_1, \ldots, h_n$ be a set of admissible heuristics. Then the heuristic $h = \max(h_1, \ldots, h_n)$ is admissible and dominates each of $h_1, \ldots, h_n$.*

In Part I, we discuss alternative ways of combining heuristic information.

Until recently, all heuristics proposed in the literature derived their information from one of four types of sources (Helmert and Domshlak, 2009):

- *Landmark heuristics* like $h^{\text{LM-cut}}$ (Helmert and Domshlak, 2009) or h^{LM} of LAMA (Richter and Westphal, 2010) compute sets of atoms or operators that have to be encountered along every plan. We discuss landmark heuristics in more detail in Section 9.1.

- *Critical path heuristics* h^m ($m \geq 1$) (Haslum and Geffner, 2000) estimate the cost of reaching a state by recursively estimating the cost of subgoals with up to m atoms. They re-occur in Section 10.5.

- *Delete relaxation heuristics* like h^+ (Hoffmann, 2005), h^{max} (Bonet and Geffner, 2001), h^{add} (Bonet and Geffner, 2001), and h^{FF} (Hoffmann and Nebel, 2001) relax the task by assuming that variables accumulate their values, and atoms that are made true by a plan, stay true. We talk about them in Section 9.2.

- *Abstraction heuristics* like pattern database heuristics (Culberson and Schaeffer, 1998; Edelkamp, 2001; Korf and Felner, 2002; Felner, Korf, and Hanan, 2004), merge-and-shrink heuristics (Helmert et al., 2014; Sievers, Wehrle, and Helmert, 2014), structural abstractions (Katz and Domshlak, 2007b, 2008b, 2009) and Cartesian abstractions (Seipp and Helmert, 2013, 2014) map the task to a simpler abstract task and use optimal abstract costs as heuristic values for the original task. Abstraction heuristics are used throughout this thesis. We formally introduce them in the following section.

The recent exception to this classification seemed to be the state equation heuristic (van den Briel et al., 2007; Bonet, 2013; Bonet and van den Briel, 2014), which we discuss in detail in Chapters 9 and 10.

2.3. Abstraction Heuristics

Abstraction heuristics relax a given planning task by mapping it to an *abstract task* that is an over-approximation of the original task. Everything that is possible in the original task is also possible in the abstract task, and so optimal plans for the abstract task cannot be more expensive than an optimal plan for the original task. We now formalize these definitions and statements, closely following Helmert, Haslum, and Hoffmann (2008).

Definition 2.8 (abstraction)**.** *Let* Π *be a planning task and* $\mathrm{TS}_\Pi = \langle \mathcal{S}, \mathcal{T}, s_\mathrm{I}, \mathrm{S}_G \rangle$ *its transition system. An* abstraction mapping *is a surjective function* $\alpha : \mathcal{S} \to \mathcal{S}^\alpha$ *that maps states to* abstract states*. A transition system* $\mathrm{TS}^\alpha_\Pi = \langle \mathcal{S}^\alpha, \mathcal{T}^\alpha, s^\alpha_\mathrm{I}, \mathrm{S}^\alpha_G \rangle$ *is an* abstraction *of* Π *for abstraction mapping* α *if*

- *the initial state is mapped to the* abstract initial state
  $s^\alpha_\mathrm{I} = \alpha(s_\mathrm{I})$*,*
- *goal states are mapped to* abstract goal states
  $\alpha(s_G) \in \mathrm{S}^\alpha_G$ *for all* $s_G \in \mathrm{S}_G$*, and*
- *for every concrete transition, there is a corresponding* abstract transition
  $\alpha(s) \xrightarrow{o} \alpha(s') \in \mathcal{T}^\alpha$ *for all* $s \xrightarrow{o} s' \in \mathcal{T}$

Given an abstraction mapping, there is a unique abstraction that has no additional goal states or transitions. The terms abstraction mapping and abstraction are therefore often used synonymously.

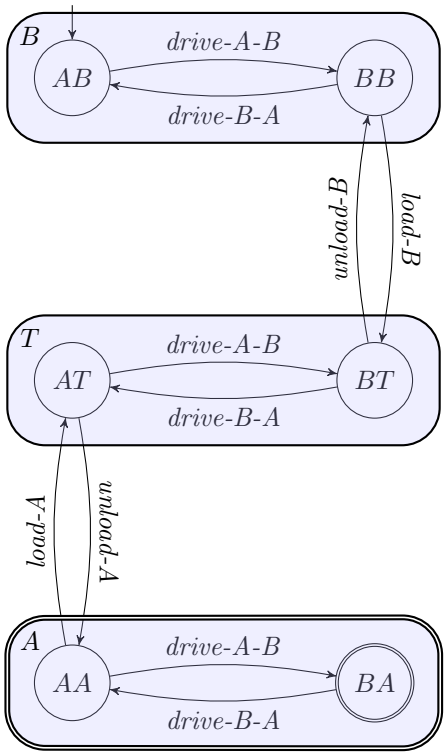
A special case of abstractions are *projections* where α projects each state to a subset of variables $P \subseteq \mathcal{V}$, called a *pattern*. That is, the abstraction mapping is $\alpha(s) = s|_P$, and the set of abstract states for a projection to P contains exactly the partial states p with $\mathit{vars}(p) = P$. Projections to a single variable are called *atomic projections*.

Figure 2.3 shows two examples for abstractions of our running example task (see Section 2.1). Figures (a) and (b) show the abstraction mappings where each state is mapped to the abstract state surrounding it. Figures (c) and (d) then show the corresponding abstract transition systems. The first abstraction (Figure 2.3a) is the atomic projection to the pattern $\{\mathit{pos}\text{-}P\}$.

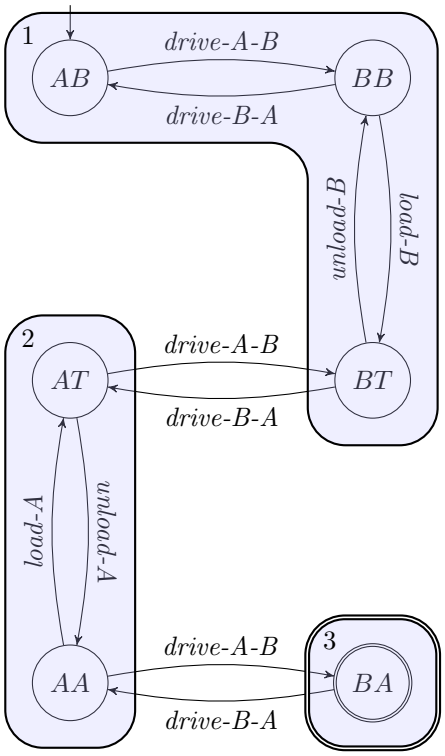
Abstractions are over-approximating the original transition system: everything that is possible in the original transition system is also possible in the abstraction. In particular, this implies that an optimal plan of Π induces a path from the abstract initial state to an abstract goal state, and since this path uses the same operators it has the same cost.

Proposition 2.2 (e.g. Helmert, Haslum, and Hoffmann, 2008)**.** *Let* Π *be a planning task with transition system* TS_Π *and* α *an abstraction of* Π *with transition system* TS_α*. For every path in* TS_Π *there is a path in* TS_α *with the same transition labels and weights.*

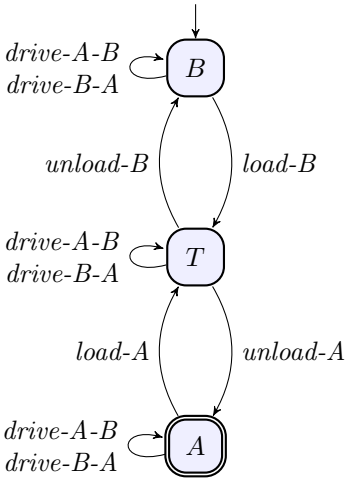
The cost of the cheapest goal path in the abstraction can therefore not exceed the optimal heuristic value and can be used as an admissible heuristic estimate.



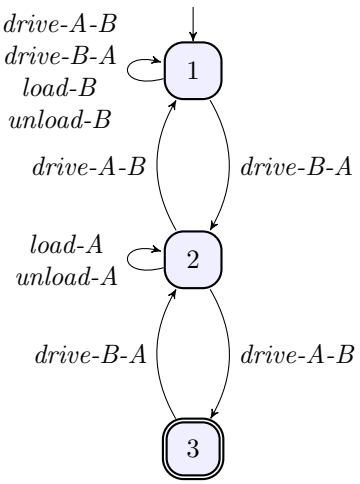
(a) Projection α_1 of the example task.



(b) Abstraction α_2 of the example task.



(c) Abstract transition system of α_1.



(d) Abstract transition system of α_2.

Figure 2.3: Two examples of abstractions and their abstract transition systems.

Definition 2.9 (abstraction heuristic)**.** *Let* Π *be a planning task and* α *an abstraction of* Π *with transition system* TS_α*. The* abstraction heuristic h^α *maps each state* s *to the cost of a shortest path from* $\alpha(s)$ *to any abstract goal state in* TS_α*.*

Proposition 2.2 shows that all abstraction heuristics are admissible (see also Pearl, 1984; Dräger, Finkbeiner, and Podelski, 2006).

Abstraction heuristics for projections are also called *pattern database* (PDB) heuristics (Culberson and Schaeffer, 1998; Edelkamp, 2001). They are usually computed by first explicitly generating the abstract transition system for the projection, then computing the abstract goal distances for all abstract states, and storing them in a table (the eponymous pattern database). During the search the heuristic value is then looked up in the pattern database.

2.4. Transition Normal Form

The discussion and theoretical analysis of many topics in this thesis is complicated for general SAS^+ tasks. Two major sources of difficulty are that there is no unique goal state and that operators can change variables on which they have no precondition. For example, we defined the value of an abstraction heuristic as the cost of the cheapest path in the abstraction that leads to *any* abstract goal state. If there were just one goal state, the qualification would no longer be necessary. Operators that mention different variables in their precondition and effect lead to similar problems. If they are applied, it is not clear how the values in a state change without knowing the state.

We now define syntactical restrictions and prove that every planning task can be efficiently converted into a task that satisfies them. Whenever an analysis is complicated by the issues mentioned above we can then assume the task satisfies the restrictions to simplify the presentation. For cases where we make this assumption, we then discuss generalizations to unrestricted SAS^+ tasks in Appendix B. In most cases the techniques are invariant under the transformation, so the simpler form can be used on the transformed tasks without a downside.

Definition 2.10 (transition normal form)**.** *A planning task* Π *is in* transition normal form *(TNF) if* $\mathit{vars}(\mathit{pre}(o)) = \mathit{vars}(\mathit{eff}(o))$ *for all operators* o *of* Π *and the goal of* Π *is a fully defined state.*

For TNF to be generally useful, we need a way to transform general planning tasks to ones in TNF that are equivalent in a formal sense. Ensuring that precondition variables also occur as effect variables is trivial: whenever this is not the case for a given precondition atom, we can add it as an effect without affecting the semantics of the planning task (this is sufficient to bring our example task into TNF). Ensuring the opposite, that effect variables V of an operator o are also precondition variables, is trickier.

A "folklore" transformation "multiplies out" variables, creating copies of o with each possible atom for V as a precondition. However, this transformation increases task size

exponentially in the worst case: if an operator has m variables occurring in the effect but not the precondition and each of these can take on n different values, the conversion produces n^m copies. We now present an alternative conversion that can only lead to a modest increase in task size.

Informally speaking, we add a new value **u** to every variable that represents that we no longer know the value of this variable. We may always "forget" the value of a variable at no cost by setting it to **u**. Operators o that have an effect but no precondition on a variable V can be extended to also require $\langle V, \mathbf{u}\rangle$. Whatever the value of V is in a state where o can be applied, we can forget it to make $\langle V, \mathbf{u}\rangle$ true. Likewise, the goal can be padded to require the value **u** for all variables that previously had no goal value.

Definition 2.11 (transition normalization)**.** *Let* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, \mathit{cost}\rangle$ *be a planning task. The* transition normalization *of* Π *is the task* $\mathit{TNF}(\Pi) = \langle \mathcal{V}', \mathcal{O}', s_{\mathrm{I}}, s'_\star, \mathit{cost}'\rangle$*, where*

- *we add a fresh value* **u** *to each variable domain:*
  $\mathcal{V}' = \{V' \mid V \in \mathcal{V}\}$ *with* $\mathit{dom}(V') = \mathit{dom}(V) \cup \{\mathbf{u}\}$*,*
- *we require* **u** *for missing variables in the goal:*
  $s'_\star = s_\star \cup \{\langle V, \mathbf{u}\rangle \mid V \in \mathcal{V} \setminus \mathit{vars}(s_\star)\}$*, and*
- *we transform each operator and add a forgetting operator for each atom:*
  $\mathcal{O}' = \{o' \mid o \in \mathcal{O}\} \cup \{\mathit{forget}_{\langle V,v\rangle} \mid \langle V, v\rangle \in \mathcal{A}\}$.
  Transformed operators
  - *keep their original cost:*
    $\mathit{cost}'(o') = \mathit{cost}(o)$*,*
  - *require* **u** *for variables missing in the precondition:*
    $\mathit{pre}(o') = \mathit{pre}(o) \cup \{\langle V, \mathbf{u}\rangle \mid V \in \mathit{vars}(\mathit{eff}(o)) \setminus \mathit{vars}(\mathit{pre}(o))\}$*, and*
  - *duplicate the precondition for variables missing in the effect:*
    $\mathit{eff}(o') = \mathit{eff}(o) \cup \{\langle V, \mathit{pre}(o)[V]\rangle \mid V \in \mathit{vars}(\mathit{pre}(o)) \setminus \mathit{vars}(\mathit{eff}(o))\}$.

  Forgetting operators
  - *are free of cost:*
    $\mathit{cost}'(\mathit{forget}_{\langle V,v\rangle}) = 0$ *and*
  - *change the value of* V *from* v *to* **u***:*
    $\mathit{pre}(\mathit{forget}_{\langle V,v\rangle}) = \{\langle V, v\rangle\}$
    $\mathit{eff}(\mathit{forget}_{\langle V,v\rangle}) = \{\langle V, \mathbf{u}\rangle\}$.

It is easy to see that $\mathit{TNF}(\Pi)$ is in transition normal form. Compared to Π, $\mathit{TNF}(\Pi)$ has $|\mathcal{V}|$ additional variable values, at most $|\mathcal{V}|$ additional goal entries, and $|\mathcal{A}|$ additional operators. The total size of an operator's preconditions and effects can at most double, since we add at most $|\mathit{eff}(o)|$ entries to the precondition of o and at most $|\mathit{pre}(o)|$ entries to its effect. With a reasonable encoding, the size of the task representation at most doubles by transition normalization. We now show that $\mathit{TNF}(\Pi)$ and Π are equivalent.

Theorem 2.1. *Let Π be a planning task. Every plan π for Π can be converted into a plan π' for TNF(Π) with the same cost in time $O(k|\pi| + |\mathcal{V}|)$, where k is the maximal number of effects of an operator. Also, every plan π' for TNF(Π) can be converted into a plan π for Π with the same cost in time $O(|\pi'|)$.*

Proof: To convert $\pi = \langle o_1, \ldots, o_n \rangle$ into π', we insert the necessary forgetting operators in front of each operator o_i: if o_i is executed in state s, insert operators to forget $\langle V, s[V] \rangle$ for all variables V on which o has an effect but no precondition. The forgetting operators are successively applicable, as at most one per variable is added. Their application reaches the state s' that is like s but has $s'[V] = \mathbf{u}$ for all variables V on which o has an effect but no precondition. Since o_i is applicable in s and has no precondition on those variables, it is applicable in s'. It affects all of those variables, so no variable has the value $\mathbf{u}$ after the application, and we have $s[\![o_i]\!] = s'[\![o'_i]\!]$. At the end of the plan, we append forgetting operators for each goal of the form $\langle V, \mathbf{u} \rangle$ in *TNF*(π). With the same argument as before, these operators are applicable and lead to $s'_\star$.

To convert a plan π' for *TNF*(Π) into a plan for Π, simply drop all forgetting operators. A forgetting operator in π' can only be required to satisfy a precondition of a transformed operator or a goal in $s'_\star \setminus s_\star$. Both do not exist in Π, so the resulting plan is applicable and leads to a goal state. □

3. Linear Programs and Mixed Integer Programs

Linear programs (LPs) and mixed integer programs (MIP) play an important role in all parts of this thesis. This chapter gives a short introduction of the most relevant concepts. Mathematical programming is a vast area of research and we only cover the basics here. For a more complete treatment of the subject we refer the reader to the literature (e.g. Schrijver, 1998).

Linear programming is the task of maximizing or minimizing a linear function subject to a set of linear constraints. For example, consider the following problem:

$$\begin{aligned} &\text{Maximize } \mathsf{X} + 2\mathsf{Y} \text{ subject to} \\ \mathsf{X} + 2\mathsf{Y} &\leq 4 \\ 2\mathsf{X} - \mathsf{Y} &\leq 2 \\ 2\mathsf{Y} &\leq 3\mathsf{X} \\ \mathsf{X} \geq 0 &\text{ and } \mathsf{Y} \geq 0 \end{aligned}$$

Figure 3.1 visualizes this example. Each constraint splits the space of possible assignments to X and Y into two half-spaces: one where the constraint is satisfied and one where it is not. Among the assignments satisfying all constraints, we are looking for one maximizing $\mathsf{X} + 2\mathsf{Y}$.

We start by introducing a standard form for linear programs and then show how more general forms of LPs can be compiled into this standard form.

Definition 3.1 (standard form)**.** *A* linear program in standard maximization form *with* n *constraints and* m *variables consists of a row vector* $\mathbf{c}$ *of* objective coefficients*, a column vector* $\mathbf{b}$ *of* bounds *and an* $n \times m$ *matrix* $\mathbf{A}$ *of* coefficients*. It is written as*

$$\begin{aligned} &\textit{Maximize } \mathbf{cx} \textit{ subject to} \\ \mathbf{Ax} &\leq \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

The inequalities are called the constraints *of the LP and the elements of* $\mathbf{x}$ *its* variables*. The* linear program in standard minimization form *for* $\mathbf{c}$*,* $\mathbf{A}$ *and* $\mathbf{b}$ *is*

$$\begin{aligned} &\textit{Minimize } \mathbf{cy} \textit{ subject to} \\ \mathbf{Ay} &\geq \mathbf{b} \\ \mathbf{y} &\geq \mathbf{0} \end{aligned}$$

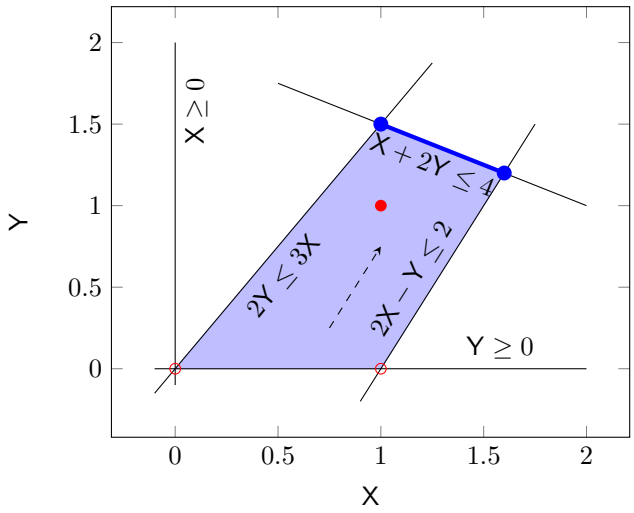


Figure 3.1: Visualization of an example LP. The blue area is the set of LP solutions. Red circles mark the MIP solutions. The dashed black arrow points in the direction of better solutions according to the objective function. The filled red circle is only optimal MIP solution. Every solution on the blue line is an optimal LP solution.

Our example LP is not in standard form but rewriting $2\mathsf{Y} \leq 3\mathsf{X}$ as $2\mathsf{Y} - 3\mathsf{X} \leq 0$ brings it into standard maximization form.

Definition 3.2 (solution)**.** *A column vector* $\mathbf{x} \in \mathbb{R}^m$ *is a* solution *of a maximization problem* P *with coefficient matrix* $\mathbf{A}$ *and bounds* $\mathbf{b}$ *if it satisfies its constraints, i.e. if* $\mathbf{Ax} \leq \mathbf{b}$ *and* $\mathbf{x} \geq \mathbf{0}$*. The* value *of a solution* $\mathbf{x}$ *is* $\mathbf{cx}$*. A solution is* optimal *if it has maximal value among all solutions.*

Analogously, solutions to minimization problems are column vectors $\mathbf{y} \in \mathbb{R}^n$ *with* $\mathbf{Ay} \geq \mathbf{b}$ *and* $\mathbf{y} \geq \mathbf{0}$*. Their value is* $\mathbf{cy}$ *and solutions with minimal value are optimal.*

An LP is feasible *if it has at least one solution. A feasible LP is* unbounded *if it has no optimal solution because for every solution, a better solution exists.*

The optimal objective value *of an LP is the value of an optimal solution.*

Figure 3.1 shows the set of solutions of our example LP in blue and the optimal solutions as a thick blue line.

In later chapters we mostly use named variables with no explicit order, which means that solutions cannot easily be written as vectors. We thus use functions that map the variable names to $\mathbb{R}$ to describe solutions. We also use SansSerifFont to distinguish LP variables from planning task variables and other notation. In the definition of our LPs we occasionally write indicator functions with Iverson brackets (Knuth, 1992) $[P]$ where $[P] = 1$ if the statement P is true and $[P] = 0$ otherwise.

Not all of the LPs we consider are in standard form but all of them can be transformed into it. For example, an equation $a\mathsf{X} = b$ is equivalent to the two inequalities $a\mathsf{X} \leq b$

and $a\mathsf{X} \geq b$. Similarly, an inequality $a\mathsf{X} \geq b$ is equivalent to $-a\mathsf{X} \leq -b$. Variables X that are not restricted to be non-negative can be expressed as the difference of two new (non-negative) variables X^+ and X^- by replacing every $a\mathsf{X}$ in a constraint with $(a\mathsf{X}^+ - a\mathsf{X}^-)$. As an example consider the following LP.

$$\begin{aligned} &\text{Maximize } \mathsf{A} - 5\mathsf{B} \text{ subject to} \\ &\mathsf{B} = 4\mathsf{A} + 7 \\ &3\mathsf{A} + 2\mathsf{B} \geq 3 \\ &\mathsf{B} \geq 0 \end{aligned}$$

This LP can be transformed into standard form by replacing A with $(\mathsf{A}^+ - \mathsf{A}^-)$ and by transforming all constraints to $\leq$-inequalities.

$$\begin{aligned} &\text{Maximize } \mathsf{A}^+ - \mathsf{A}^- - 5\mathsf{B} \text{ subject to} \\ -4\mathsf{A}^+ + 4\mathsf{A}^- + \mathsf{B} &\leq 7 \\ 4\mathsf{A}^+ - 4\mathsf{A}^- - \mathsf{B} &\leq -7 \\ -3\mathsf{A}^+ + 3\mathsf{A}^- - 2\mathsf{B} &\leq -3 \\ \mathsf{A}^+ \geq 0, \mathsf{A}^- \geq 0, \mathsf{B} &\geq 0 \end{aligned}$$

The standard form LP consists of the following components:

$$\mathbf{A} = \begin{pmatrix} -4 & 4 & 1 \\ 4 & -4 & -1 \\ -3 & 3 & -2 \end{pmatrix} \quad \mathbf{x} = \begin{pmatrix} \mathsf{A}^+ \\ \mathsf{A}^- \\ \mathsf{B} \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} 7 \\ -7 \\ -3 \end{pmatrix} \quad \mathbf{c} = (1, -1, -5)$$

An optimal solution of the LP is f with $f(\mathsf{A}^+) = 1$, $f(\mathsf{A}^-) = 2$, $f(\mathsf{B}) = 3$, which corresponds to the solution f' with $f'(\mathsf{A}) = -1$ and $f'(\mathsf{B}) = 3$ in the original problem.

An important concept in linear programming is the *dual* of an LP. Dualization offers a new perspective on an LP where the role of variables and constraints is reversed. As we will see, the dual and its original problem (called the *primal*) are closely linked.

Definition 3.3 (dual)**.** *The* dual *of the maximization LP with n constraints and m variables*

$$\begin{aligned} &\textit{Maximize } \mathbf{cx} \textit{ subject to} \\ &\mathbf{Ax} \leq \mathbf{b} \\ &\mathbf{x} \geq \mathbf{0} \end{aligned}$$

is the minimization LP with m constraints and n variables

$$\begin{aligned} &\textit{Minimize } \mathbf{b}^\top\mathbf{y} \textit{ subject to} \\ &\mathbf{A}^\top\mathbf{y} \geq \mathbf{c}^\top \\ &\mathbf{y} \geq \mathbf{0} \end{aligned}$$

The dual of a minimization LP is defined analogously, so that the dual of the dual is the primal again.

Linear programs with unrestricted variables and equations can be dualized by bringing them into standard form first. This is slightly cumbersome and makes it hard to keep track of the original variables, though. Since both unrestricted variables and equations are used in this thesis, we show how to directly generate the dual of an LP containing them. Consider the following maximization LP where the last $m - k$ constraints are equations and the last $n - l$ variables are unrestricted:

$$\text{Maximize } \mathbf{cx} \text{ subject to}$$

$$\begin{aligned}
\sum_{j=1}^{n} a_{ij} x_j &\leq b_i && \text{for } 1 \leq i \leq k \\
\sum_{j=1}^{n} a_{ij} x_j &= b_i && \text{for } k < i \leq m \\
x_j &\geq 0 && \text{for } 1 \leq j \leq l
\end{aligned}$$

In the dual, constraints that correspond to unrestricted primal variables are equations and dual variables that correspond to primal equations are unrestricted:

$$\text{Minimize } \mathbf{b}^\top \mathbf{y} \text{ subject to}$$

$$\begin{aligned}
\sum_{i=1}^{m} a_{ji} y_i &\geq c_j && \text{for } 1 \leq j \leq l \\
\sum_{i=1}^{m} a_{ji} y_i &= c_j && \text{for } l < j \leq n \\
y_i &\geq 0 && \text{for } 1 \leq i \leq k
\end{aligned}$$

It is easy to prove that the LPs are each others dual by converting them to normal form, generating the dual, and reintroducing unrestricted variables and equations.

A central theorem about linear programs states that a bounded feasible LP and its dual have the same optimal objective value. This is called *strong duality*.

Theorem 3.1 (e.g. Schrijver, 1998)**.** *Let P be a linear program and P' its dual. If P is bounded feasible then so is P' and the optimal objective values of P and P' are equal. If P is unbounded then P' is infeasible. If P is infeasible then P' in unbounded or infeasible.*

In addition to LPs, we use *mixed integer programs* (MIPs) that restrict some of the variables to take only integer values. If both variables in the example shown in Figure 3.1 are restricted to integers, only three solutions remain (shown in red) and only one of them is optimal. The LP that is created by dropping all integer restrictions in a MIP is called the *LP relaxation* of the MIP. Every solution of a MIP also is a solution of its LP relaxation, so the optimal objective value can only go up if the MIP maximizes or down if the MIP minimizes.

Finding the optimal solution of an LP is possible with polynomial methods (e.g. Khachiyan, 1980; Karmarkar, 1984) but worst-case exponential algorithms based on the simplex algorithm (Dantzig, 1951) are often better in practice. Solving MIPs on the other hand is NP-complete (Schrijver, 1998). Highly efficient MIP solvers exist, but solving a MIP is still significantly harder than solving an LP in practice. In fact, most MIP solvers solve the LP relaxation of the MIP as a first step.

Part I.

Cost Partitioning

4. Introduction to Cost Partitioning

A set of admissible heuristics is called *additive* if their sum is admissible. A famous early example of additive heuristics are *disjoint pattern database heuristics* for the sliding-tile puzzle (Korf and Felner, 2002), which were later generalized to classical planning by Edelkamp (2001). Disjoint pattern databases use a simple sufficient condition for additivity: a set of patterns (a *pattern collection*) is additive if no operator affects more than one pattern. In this case, operators have a non-empty effect in at most one of the projections. An optimal plan can be seen as interleaved plans for all projections. The total cost of cheapest plans for all projections is thus an admissible estimate.

In our example task the two projections to the position of the truck and the position of the package are additive since driving only affects the truck and (un)loading only affects the packages.

Haslum et al. (2007) extend the idea of additive pattern databases by introducing the *canonical heuristic*. A pattern collection $\mathcal{C}$ is only additive if all patterns in $\mathcal{C}$ are pairwise disjoint. If this is not the case, then the sum over the respective projection heuristics is not admissible. However, a subset of $\mathcal{C}$ could be additive, and it is possible to compute such subsets in advance. Let $\mathit{MAS}(\mathcal{C})$ denote all maximal (w.r.t. set inclusion) additive subsets of $\mathcal{C}$. Then the *canonical heuristic* value $h^{\text{canon}}_{\mathcal{C}}$ is defined as

$$h^{\text{canon}}_{\mathcal{C}}(s) = \max_{A \in \mathit{MAS}(\mathcal{C})} \sum_{P \in A} h^P(s).$$

The computation of $\mathit{MAS}(\mathcal{C})$ involves finding maximal cliques in the graph of heuristics with edges between each pair of additive heuristics. This is an NP-complete problem, but it only has to be solved once which makes this heuristic practical for sufficiently small pattern collections. Haslum et al. also introduce the iPDB heuristic which computes the canonical heuristic over a pattern collection discovered by a local search and remains a state-of-the-art heuristic (Scherrer, Pommerening, and Wehrle, 2015).

Haslum, Bonet, and Geffner (2005) use a broader definition and generalize the notion of additivity from PDB heuristics to arbitrary heuristics. They partition operators into disjoint sets $\mathcal{O}_1, \ldots, \mathcal{O}_n$ and define $h_{\mathcal{O}_i}$ as computing heuristic h but ignoring the cost of all operators not in $\mathcal{O}_i$. The heuristics $h_{\mathcal{O}_i}$ for $1 \leq i \leq n$ are then additive because the contribution of all operators in a plan can be partitioned into the same groups as the operators. This opens up a way to make any set of heuristics admissible by assigning each operator to one of the heuristics. Haslum, Bonet, and Geffner apply this to critical path and PDB heuristics.

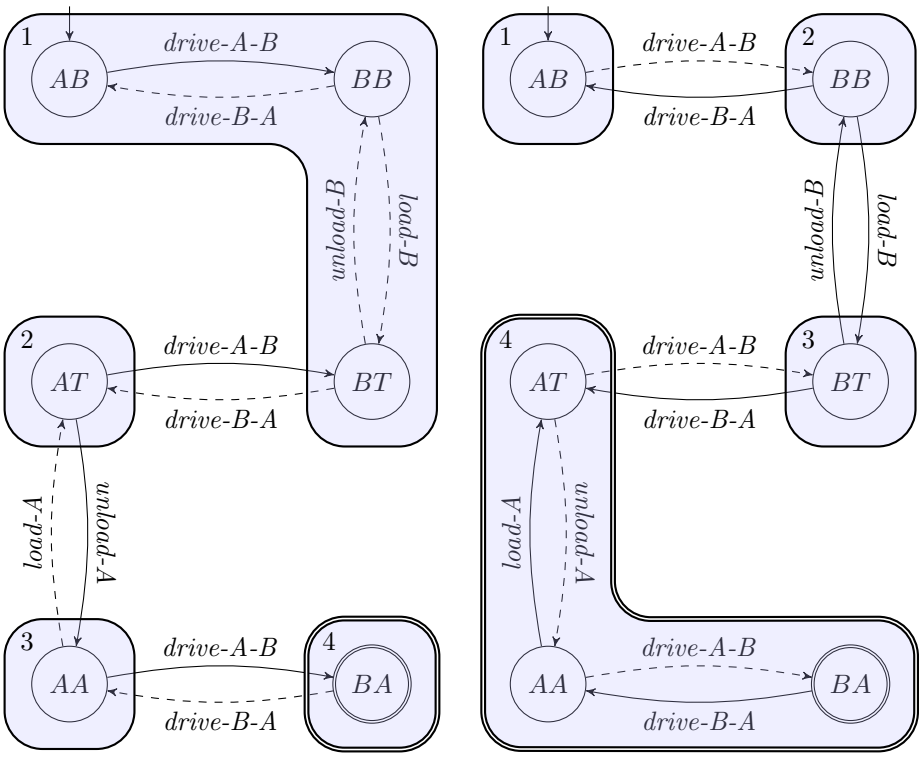


Figure 4.1: Partitioning the operators between two abstractions of the example task. If only the contributions of *unload-A* and *drive-A-B* are counted in the first abstraction and only the contributions of the remaining operators in the second abstraction, the sum of heuristic values is admissible. Without partitioning, the sum would be inadmissible.

Consider our running example task together with the abstractions shown in Figure 4.1. Each abstraction heuristic has a heuristic value of 3 and the sum of both heuristics is not admissible (the optimal plan cost is 5). The maximum of both heuristics is admissible, but in this case, we can achieve a higher heuristic value by partitioning the operators into two sets: If we only count the contributions of *unload-A* and *drive-A-B* in the first abstraction and ignore them in the second abstraction, both heuristic values are 2. Their sum is an admissible estimate that exceeds the maximum of both heuristics.

All techniques discussed so far can be seen in the framework of Haslum, Bonet, and Geffner (2005) as partitioning the operators into disjoint sets $\mathcal{O}_1, \ldots, \mathcal{O}_n$ and computing the heuristic value $\sum_i h_{\mathcal{O}_i}$. *Cost partitioning* (Katz and Domshlak, 2007a, 2008a; Yang et al., 2008; Katz and Domshlak, 2010b) generalizes this idea. Instead of letting each operator contribute to one of the heuristics with its full cost, cost partitioning may split the cost of an operator between heuristics so an operator may contribute fractions of its actual cost to all heuristics. The total cost remains admissible if the costs are correctly *partitioned* between the heuristics.

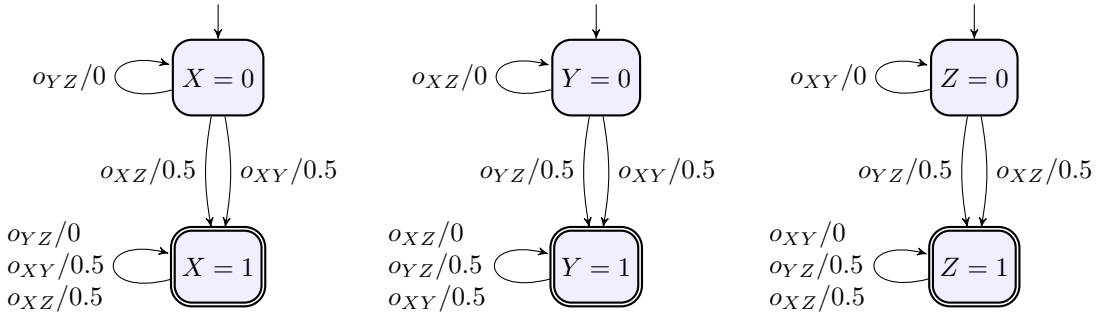


Figure 4.2: Atomic projections of a task where assigning the full cost of each operator to one of the abstractions is not optimal. The edge labels show an optimal cost partitioning.

Definition 4.1 (non-negative operator cost partitioning)**.** *Let Π be a planning task with operators $\mathcal{O}$ and cost function cost. A* non-negative operator cost partitioning *for Π is a tuple $\langle cost_1, \ldots, cost_n \rangle$ where $cost_i : \mathcal{O} \to \mathbb{R}_0^+$ for $1 \leq i \leq n$ and*

$$\sum_{i=1}^{n} cost_i(o) \leq cost(o) \text{ for all } o \in \mathcal{O}.$$

We call the condition $\sum_{i=1}^{n} cost_i(o) \leq cost(o)$ the *cost partitioning property*. It ensures that admissible heuristics are additive under the cost functions $cost_i$.

Proposition 4.1 (Katz and Domshlak, 2010b)**.** *Let Π be a planning task, let $h_1, \ldots, h_n$ be admissible heuristics for Π, and let $P = \langle cost_1, \ldots, cost_n \rangle$ be a non-negative operator cost partitioning for Π. Then $h_P(h_1, \ldots, h_n, s) = \sum_{i=1}^{n} h_i(s, cost_i)$ is an admissible heuristic estimate for state s.*

Consider the following example task over three binary variables X, Y, and Z (Bonet and Helmert, 2010). All variables are initially 0 and should be 1 in the goal state. There are three operators o_{XY}, o_{XZ}, and o_{YZ} with cost 1. Operator o_{XY} sets X and Y to 1 with no precondition, and the other operators are defined analogously. Figure 4.2 shows the atomic projections of the task (ignore the numbers after the operators for now).

In the framework of Haslum, Bonet, and Geffner (2005), we compute the heuristic $h^X_{\mathcal{O}_X} + h^Y_{\mathcal{O}_Y} + h^Z_{\mathcal{O}_Z}$ for some disjoint subsets $\mathcal{O}_X, \mathcal{O}_Y, \mathcal{O}_Z$ of the operators. The heuristic value of $h^X_{\mathcal{O}_X}$ can only be positive if $\mathcal{O}_X$ contains o_{XY} and o_{XZ}. Likewise, $h^Y_{\mathcal{O}_Y}$ and $h^Z_{\mathcal{O}_Z}$ can only become positive if $\mathcal{O}_Y$ and $\mathcal{O}_Z$ contain at least two operators. Since the sets have to be disjoint only one of the heuristics can have a non-zero heuristic value. No matter how the operators are partitioned, the total heuristic value can not exceed the maximum over the projection heuristics.

The edge labels in Figure 4.2 show a *uniform cost partitioning* (e.g. Karpas and Domshlak, 2009) that partitions the cost of each operator into equal parts for each abstraction where the operator is relevant. In the example each operator is assigned a cost

of 0.5 in two of the abstractions. The heuristic value of all three abstraction heuristics is 0.5 and their sum, 1.5, is admissible. There is no way to distribute the costs that leads to a higher heuristic value, so uniform cost partitioning is *optimal* for the example, even though this is not always the case.

Definition 4.2 (optimal non-negative operator cost partitioning)**.** *Let Π be a planning task and let $h_1, \ldots, h_n$ be admissible heuristics for Π. Let $\mathcal{P}^+$ be the set of all non-negative operator cost partitionings. A partitioning $P^* \in \mathcal{P}^+$ is* optimal *in state s if*

$$P^* \in \arg\max_{P \in \mathcal{P}^+} h_P(h_1, \ldots, h_n, s).$$

The optimal non-negative operator cost partitioning heuristic $h^{\text{OCP+}}$ *for $h_1, \ldots, h_n$ is the function*

$$h^{\text{OCP+}}_{\{h_i | 1 \leq i \leq n\}}(s) = \max_{P \in \mathcal{P}^+} h_P(h_1, \ldots, h_n, s).$$

Maximization can be seen as an extreme case of cost partitioning, where the full costs of all operators are attributed to one heuristic, and operator costs of 0 are used for all other heuristics. The example above shows that maximization is not always optimal. One aspect that is overlooked when combining heuristic estimates with maximization is that some of them might be additive.

Like maximization, the canonical heuristic can also be seen as a case of cost partitioning. Let $A^{\max}$ be the maximizing element of $\mathit{MAS}(\mathcal{C})$. Then the full cost of each operator is counted in the projection to the pattern in $A^{\max}$ that it affects. This can only be one pattern in $A^{\max}$ because the elements of $A^{\max}$ are additive. All other costs are set to 0. In each projection, all operators that affect it have their full cost, so the projection heuristics have the same value as under the original cost function.

Every pattern P occurs in at least one additive set of $\mathit{MAS}(\mathcal{C})$ because $\{P\}$ is an additive set and adding more additive patterns from $\mathcal{C}$ can only make the set larger. In particular, there is a set in $\mathit{MAS}(\mathcal{C})$ that includes a pattern P for which h^P is maximal, so the canonical heuristic for a collection $\mathcal{C}$ dominates the maximum over the projection heuristics for patterns in $\mathcal{C}$.

However, combining the projections with the canonical heuristic is still not guaranteed to be optimal. The example in Figure 4.2 shows that cost partitioning can yield higher heuristic values than the maximum, even if the component heuristics are not additive.

Among other contributions, Katz and Domshlak (2010b) showed how to compute an optimal operator cost partitioning for a given state and a wide class of abstraction heuristics in polynomial time, and we repeat the main idea here for completeness.

The abstraction heuristic for an abstraction α under a cost function cost^α can be described using linear constraints. Let $\mathrm{TS}^\alpha = \langle \mathcal{S}^\alpha, \mathcal{T}^\alpha, s^\alpha_{\mathrm{I}}, \mathrm{S}^\alpha_G \rangle$ be the transition system of α. We then use one LP variable H^α_s for each abstract state $s \in \mathcal{S}^\alpha$ to represent the heuristic value of all original states that are mapped to s by α. The value of H^α_s

is defined as the cost of a shortest path according to $cost^\alpha$ in TS^α. There are well-known linear programming models that express the shortest path problem as a set of linear inequalities (Goldfarb, Hao, and Kai, 1990). For example, the heuristic value $h^\alpha(s, cost^\alpha)$ is the optimal objective value of the following linear program.

$$\text{Maximize } \mathsf{H}^\alpha_{\alpha(s)} \text{ subject to}$$

$$\mathsf{H}^\alpha_{s_G} = 0 \qquad \text{for all } s_G \in \mathrm{S}^\alpha_G \tag{4.1}$$

$$\mathsf{H}^\alpha_{s'} \leq \mathsf{H}^\alpha_{s''} + cost^\alpha(o) \qquad \text{for all } s' \xrightarrow{o} s'' \in \mathcal{T}^\alpha \text{ if } s' \text{ and } s'' \text{ are alive} \tag{4.2}$$

$$\mathsf{H}^\alpha_{s'} \geq 0 \qquad \text{for all } s' \in \mathcal{S}^\alpha \tag{4.3}$$

The model only contains constraints for transitions between alive states, which are the transitions relevant for the shortest path. This is important to note because other formalizations often have constraints for all transitions to simplify the model. If all operator costs are non-negative, these additional constraints are redundant, i.e. they can always be satisfied without influencing the optimal objective value. To see this, note that dead states are either unreachable or cannot reach the goal. Assigning the true goal distance to all states that can reach the goal and ∞ to all other dead states satisfies the constraints for transitions adjacent to them. We explicitly exclude these redundant constraints here because they are no longer redundant under general cost functions, which we discuss in the following chapter.

Katz and Domshlak (2010b) optimize the cost partitioning over a set of abstractions A and introduce additional LP variables C^α_o for each $\alpha \in A$ and each $o \in \mathcal{O}$ to represent $cost^\alpha(o)$. The cost partitioning property is a linear constraint over these variables. Together with constraints (4.1)–(4.3) for each abstraction $\alpha \in A$ we get a linear program with the optimal objective value $h^{\mathrm{OCP+}}(s)$.

$$\text{Maximize } \sum_{\alpha \in A} \mathsf{H}^\alpha_{\alpha(s)} \text{ subject to}$$

$$\mathsf{H}^\alpha_{s_G} = 0 \qquad \text{for all } \alpha \in A \text{ and } s_G \in \mathrm{S}^\alpha_G$$

$$\mathsf{H}^\alpha_{s'} \leq \mathsf{H}^\alpha_{s''} + \mathsf{C}^\alpha_o \qquad \text{for all } \alpha \in A \text{ and } s' \xrightarrow{o} s'' \in \mathcal{T}^\alpha \text{ if } \alpha(s') \text{ and } \alpha(s'') \text{ are alive}$$

$$\sum_{\alpha \in A} \mathsf{C}^\alpha_o \leq cost(o) \qquad \text{for all } o \in \mathcal{O}$$

$$\mathsf{H}^\alpha_{s'} \geq 0 \qquad \text{for all } \alpha \in A \text{ and } s' \in \mathcal{S}^\alpha$$

$$\mathsf{C}^\alpha_o \geq 0 \qquad \text{for all } \alpha \in A \text{ and } o \in \mathcal{O}$$

This LP contains $|\mathcal{O}||A| + \sum_{\alpha \in A} |\mathcal{S}_\alpha|$ variables and $O(\sum_{\alpha \in A} |\mathrm{TS}_\alpha|)$ constraints. If the transition systems of all abstractions are small enough (i.e. if $|\mathrm{TS}_\alpha|$ is polynomial in the size of Π) then $h^{\mathrm{OCP+}}$ can be computed by solving the above LP. However, for

reasons of efficiency, non-optimal cost partitioning such as uniform cost partitioning, *zero-one cost partitioning* (e.g. Edelkamp, 2006), or saturated cost partitioning (Seipp and Helmert, 2014) is also commonly used.

Planning tasks can be abstracted in a way that the abstraction is tractable but still has exponentially many abstract states. One example are *fork decompositions* (Katz and Domshlak, 2010a), which can be computed in polynomial time by representing the abstraction implicitly. The LP we discussed requires an explicit transition system but Katz and Domshlak show cases where $h^{\text{OCP+}}$ can be computed in polynomial time for implicit abstractions as well.

In the following chapter we introduce two extensions to non-negative operator cost partitioning that can derive higher heuristic values from the same set of abstractions. Both extensions are independent and can be combined for even higher heuristic values. The extensions are also independent of the way abstractions are represented, but in our experiments and Parts II and III we focus on explicit state abstractions.

5. Extensions to Cost Partitioning

In the following sections, we discuss two cases where non-negative operator cost partitioning misses information contained in the abstractions. In each case, we propose a generalization of cost partitioning that can exploit this information to achieve higher heuristic estimates. The two extensions are independent of each other, and using both of them can lead to even higher heuristic values.

5.1. General Operator Cost Partitioning

Consider the two abstractions of our example task shown in Figure 5.1 (ignore the numbers next to the operators for now). It is easy to see that the heuristic value of the first abstraction is determined by the amount of $\mathit{cost}(\mathit{drive\text{-}A\text{-}B})$ assigned to this abstraction. Likewise, the heuristic value of the second abstraction is determined by the distribution of the cost of *load-B*. With non-negative operator cost partitioning, both heuristic values can be at most 1. An optimal partitioning assigns the full cost of *drive-A-B* to the first abstraction and the full cost of *load-B* to the second. The combined heuristic value is $h^{\text{OCP+}}(s_{\text{I}}) = 2$.

However, there is more information contained in the abstractions: as we can see from the second abstraction, the package has to be loaded at location B, i.e. the operator *load-B* has to be used. The first abstraction shows that at least three *drive* operators are necessary to load the package at B and reach the goal state afterwards, e.g. with the abstract plan $\langle \mathit{drive\text{-}A\text{-}B}, \mathit{load\text{-}B}, \mathit{drive\text{-}B\text{-}A}, \mathit{drive\text{-}A\text{-}B} \rangle$. Under non-negative cost partitioning this plan is only optimal in the first abstraction if its cost is zero. Otherwise, the plan $\langle \mathit{drive\text{-}A\text{-}B} \rangle$ is always cheaper. We now show how cost partitioning can be used to recognize that using *load-B* causes additional cost in the first abstraction and to account for this cost in the combined heuristic value.

Up until now, all cost functions were restricted to non-negative values. Other than the intuitive reason that the planning task's cost function is non-negative, there is no compelling reason to do so. We now remove this restriction and permit general (possibly negative) cost functions $\mathit{cost} : \mathcal{O} \to \mathbb{R}$. This means that we must generalize some concepts and notations that relate to operator costs. Shortest paths in weighted digraphs that permit negative weights are well-defined unless there is a cycle of negative overall cost that is incident to a path from the source state to the goal. We can retain our definition of optimal plans except for this case, in which plans of arbitrarily low cost exist and we set $h^*(s, \mathit{cost}') = -\infty$. Heuristic estimates may now necessarily take negative

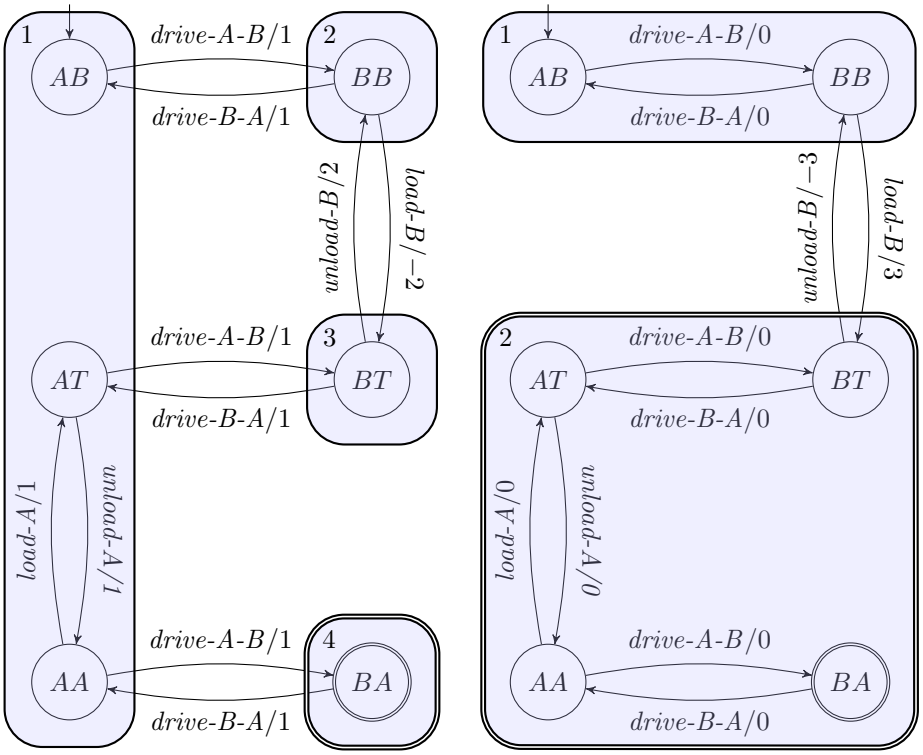


Figure 5.1: General operator cost partitioning over two abstractions of the example task.

values or even $-\infty$ to be admissible. The definition of operator cost partitioning can be generalized in a straight-forward way:

Definition 5.1 (General operator cost partitioning)**.** *Let Π be a planning task with operators $\mathcal{O}$ and cost function cost. A* general operator cost partitioning *for Π is a tuple $\langle cost_1, \ldots, cost_n \rangle$ where $cost_i : \mathcal{O} \to \mathbb{R}$ for $1 \leq i \leq n$ and*

$$\sum_{i=1}^{n} cost_i(o) \leq cost(o) \text{ for all } o \in \mathcal{O}.$$

Heuristic estimates remain admissible when considering general cost partitionings and we can show a theorem analogous to Proposition 4.1.

Theorem 5.1. *Let Π be a planning task, let $h_1, \ldots, h_n$ be admissible heuristics for Π, and let $P = \langle cost_1, \ldots, cost_n \rangle$ be a general operator cost partitioning for Π. Then $h_P(h_1, \ldots, h_n, s) = \sum_{i=1}^{n} h_i(s, cost_i)$ is an admissible heuristic estimate for s. If any term in the sum is ∞, the sum is defined as ∞, even if another term is $-\infty$.*

Proof: Consider an arbitrary state s. If Π has no s-plan then $h^*(s) = \infty$ and every estimate is admissible.

We are left with the case where Π has an s-plan. Then all $h_i(s, \mathit{cost}_i)$ are finite or $-\infty$ because an admissible heuristic can only produce ∞ for states without plans, no matter what the cost function is. Consider the case where $h^*(s) > -\infty$. Let $\pi = \langle o_1, \ldots, o_k \rangle$ be an optimal s-plan for Π. We get:

$$\begin{aligned}
h_P(h_1, \ldots, h_n, s) &= \sum_{i=1}^{n} h_i(s, \mathit{cost}_i) && \text{(Definition of } h_P\text{)} \\
&\leq \sum_{i=1}^{n} h^*(s, \mathit{cost}_i) && \text{(Admissibility of } h_i\text{)} \\
&\leq \sum_{i=1}^{n} \sum_{j=1}^{k} \mathit{cost}_i(o_j) && \text{(Optimality of } h^*\text{)} \\
&= \sum_{j=1}^{k} \sum_{i=1}^{n} \mathit{cost}_i(o_j) && \text{(Basic arithmetic)} \\
&\leq \sum_{j=1}^{k} \mathit{cost}(o_j) && \text{(Cost partitioning property)} \\
&= h^*(s). && \text{(Definition of } h^*\text{)}
\end{aligned}$$

In the case where $h^*(s) = -\infty$, there exists a state s' on a path from s to a goal state with an incident negative-cost cycle π', i.e. π' with $s'[\![\pi']\!] = s'$ and $\mathit{cost}(\pi') < 0$. From the cost partitioning property, we get $\sum_{i=1}^{n} \mathit{cost}_i(\pi') \leq \mathit{cost}(\pi') < 0$, and hence $\mathit{cost}_j(\pi') < 0$ for at least one $j \in \{1, \ldots, n\}$. This implies $h^*(s, \mathit{cost}_j) = -\infty$ and hence $h_P(h_1, \ldots, h_n, s) = -\infty$, concluding the proof. □

The notion of an optimal cost partitioning and the corresponding heuristic can be extended in a straight-forward way:

Definition 5.2 (optimal general operator cost partitioning)**.** *Let Π be a planning task and let $h_1, \ldots, h_n$ be admissible heuristics for Π. Let $\mathcal{P}$ be the set of all general operator cost partitionings. A partitioning $P^* \in \mathcal{P}$ is* optimal *in state s if*

$$P^* \in \arg\max_{P \in \mathcal{P}} h_P(h_1, \ldots, h_n, s).$$

The optimal general operator cost partitioning heuristic h^{OCP} *for $h_1, \ldots, h_n$ is the function*

$$h^{\text{OCP}}_{\{h_i | 1 \leq i \leq n\}}(s) = \max_{P \in \mathcal{P}} h_P(h_1, \ldots, h_n, s).$$

Since every non-negative operator cost partitioning is also a general partitioning, h^{OCP} dominates $h^{\text{OCP+}}$. The example in Figure 5.1 demonstrates that the dominance

is strict. The numbers next to the operators show an optimal general operator cost partitioning. Assigning a cost of -2 to *load-B* in the first abstraction does not influence the heuristic value as long as it is not part of a negative cost cycle. The cost of *load-B* can then be set to 3 in the second abstraction because $3 - 2 \leq 1$. This raises the heuristic value of the abstraction to 3. Note that the heuristic value remains admissible. Setting the cost of *load-B* even higher in the second abstraction and lower in the first creates a negative cost cycle in the first abstraction. This still is an admissible estimate because the first abstraction then contributes a heuristic value of $-\infty$, reducing the total heuristic to $-\infty$ as well.

Note that by allowing negative costs, heuristic values of individual heuristics can become negative. However, we do not have to make special consideration for this in the search algorithm. Since all operator costs in the planning task are non-negative, the heuristic $h_0(s) = 0$ is always admissible, and we can use the admissible heuristic $h'(s) = \max(h(s), h_0(s))$ instead of any admissible heuristic h for the search. In the context of a cost partitioning, though, a negative contribution of one of the component heuristics may be necessary to ensure admissibility, so component heuristics may not be individually maximized with h_0 and may not be left out of the sum.

Let us consider cost partitioning over projections to demonstrate the benefits of general cost partitioning. Not all projections are useful for non-negative cost partitioning. A projection to a pattern that does not include a goal variable, for example, has a heuristic value of 0 under every non-negative cost partitioning. Assigning non-zero operator costs in such a projection can never be necessary for an optimal non-negative cost partitioning. Likewise, the projection to two causally unrelated variables does not contain more information than the two atomic projections together. Formally, this can be described with the notion of an *interesting pattern* (Pommerening, Röger, and Helmert, 2013). If a pattern is not interesting, there is a set of additive smaller patterns whose projections provide the same heuristic values. Optimal non-negative cost partitioning heuristics cannot benefit from projections to "uninteresting" patterns. This is no longer the case if we consider general cost partitioning.

Figure 5.2 shows an example where considering an uninteresting pattern increases the heuristic value with general cost partitioning. The projection to V_2 is not interesting in this example according to the definition of Pommerening, Röger, and Helmert because V_2 is not a goal variable. However, an optimal general cost partitioning over both projections, as the one shown in the figure, achieves the perfect heuristic value of 2, whereas considering only the "interesting" abstraction would limit the heuristic value to 1.

Projections to non-goal variables are not the only case where general costs increase the heuristic value. Figure 5.3 shows a family of tasks similar to the previous example but with a goal value for V_2. The tasks model two-digit counters in base $n + 1$. One type of operator can increase the last digit by one, another can increase the second digit when the first has the value n (which resets the first digit to 0). The numbers in the figure show a general cost partitioning between the two atomic projections. Using

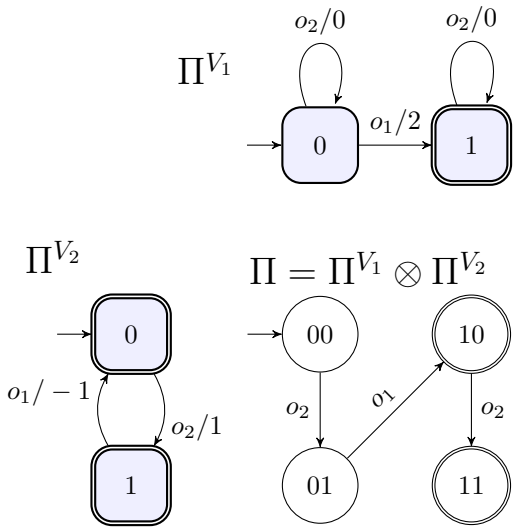


Figure 5.2: Example task Π with binary variables V_1 and V_2. Above and to the left of the original task we show the projections to V_1 and V_2. General cost partitioning benefits from the projection to V_2 even though V_2 is not mentioned in the goal.

general costs we can make the "carry"-operator more expensive in the projection to V_1 and reach the perfect heuristic value of $(n+1)^2 - 1$. This is interesting because both projections only have $n+1$ states. In a projection with $n+1$ abstract states any non-negative cost partitioning can achieve a heuristic value of at most n if all operators originally have cost 1. In such tasks the heuristic value of non-negative cost partitioning over atomic projections is limited by the number of atoms ($2n+2$ in the example task). With general costs it is possible to exceed this limit.

Korf (1997) conjectured that the time taken by an IDA* search is inversely related to the memory used for a PDB heuristic in a given search space. Holte (2013) discusses the implications for abstraction heuristics. Since search spaces scale exponentially, Korf's conjecture implies that exponentially larger PDBs are necessary to keep the time constant. Holte suggest using multiple PDBs as a way around this limit. The previous example can be extended from a two-digit counter to an n-digit counter to demonstrate that n projections with n states each can accurately represent a search space with n^n states.

Consider the task Π in Figure 5.4 as a final example for the advantages of general cost partitioning. The task is unsolvable even though an abstract goal state is reachable in both abstractions. Non-negative cost partitioning can only return an infinite value if no abstract goal state is reachable in one of the projections, so it would not detect that Π is unsolvable. The labels next to the edges show a general cost partitioning for any value of M. In the projection to V_1 the cost of the only operator is M and the operator is needed to reach the abstract goal. In the projection to V_2 the cost is $-M$ but

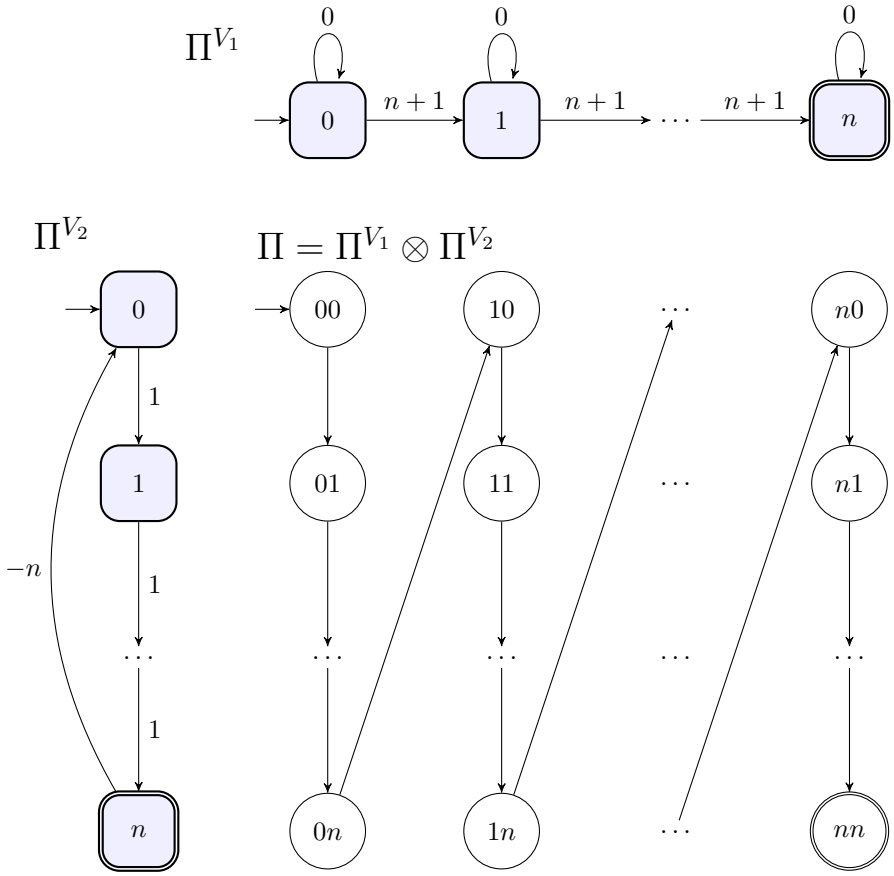


Figure 5.3: Family of planning tasks where general cost partitioning over two atomic projections of a unit-cost task produces higher estimates than the number of abstract states in the projections.

the operator is not part of any abstract plan. The heuristic value of this projection is 0 under all cost functions since the initial state of the projection to V_2 is also an abstract goal state. The cost-partitioned heuristic value is thus $M + 0 = M$. Since the value of M can be arbitrarily large, the heuristic h^{OCP} has the value ∞. Thus, general operator cost partitioning is able to detect that Π is unsolvable.

In Chapter 4 we introduced the LP by Katz and Domshlak (2010b) to compute an optimal operator cost partitioning for abstractions. We explicitly avoided adding constraints for transitions adjacent to dead states. For non-negative cost functions this is not necessary because every solution for the alive part of the transition system can be extended to the dead states without violating the additional constraints. The previous example demonstrates that such constraints are too restrictive in the context of general cost functions. In the projection to V_2, the only operator induces a self-loop on the state 1. The constraint for this transition is $\mathsf{C}_o^\alpha \geq 0$ and rules out the optimal cost partitioning. However, we can ignore it because the state is unreachable.

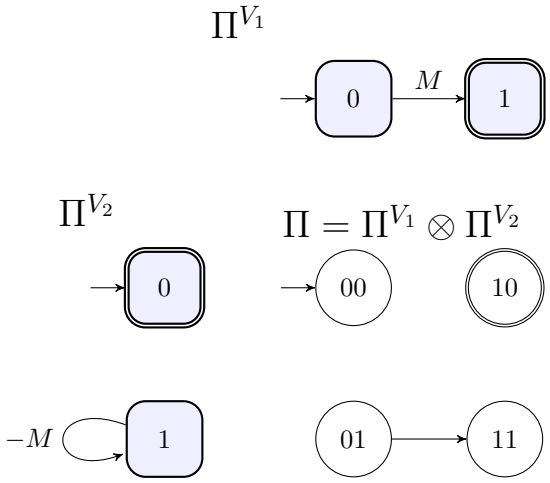


Figure 5.4: Example task Π with binary variables V_1 and V_2. Above and to the left of the original task we show the projections to V_1 and V_2. General cost partitioning detects that the task is unsolvable even though goal states are reachable in both abstractions.

5.2. Non-negative Transition Cost Partitioning

We now turn to a different generalization of operator cost partitioning that is orthogonal to using general cost functions. We introduce it in the context of non-negative cost functions first and discuss the combination with general cost functions in the following section. As an example, consider the two abstractions in Figure 5.5 and ignore the numbers for now. Both abstractions have an optimal plan $\langle$*drive-A-B*$\rangle$ under any non-negative cost function. We can split the cost of *drive-A-B* between the two abstractions with operator cost partitioning but since the sum cannot exceed the original cost of 1, optimal operator cost partitioning cannot produce a value higher than 1.

However, in this case we can see that the operator *drive-A-B* is used in different contexts in the two abstractions. We can replace the operator by three operators *drive-A-B*$_A$, *drive-A-B*$_T$, and *drive-A-B*$_B$, where *drive-A-B*$_x$ is like *drive-A-B* but with the additional precondition $\langle$*pos-P*$, x\rangle$. An optimal operator cost partitioning then assigns the full cost of *drive-A-B*$_B$ to the first abstraction and the full cost of *drive-A-B*$_A$ to the second abstraction (and a cost of 0 in the other abstraction). This brings the total heuristic value up to 2.

Replacing an operator with restricted copies for different situations is called *context splitting* (Röger, Pommerening, and Helmert, 2014). Formally defining context splitting is easier for the more general ADL planning tasks, where operator preconditions can be arbitrary propositional formulas over atoms instead of partial states. We define context splitting for such tasks and then show how a special case of it applies to SAS$^+$ tasks.

Definition 5.3 (context splitting)**.** *Let Π be an ADL planning task and o one of its op-*

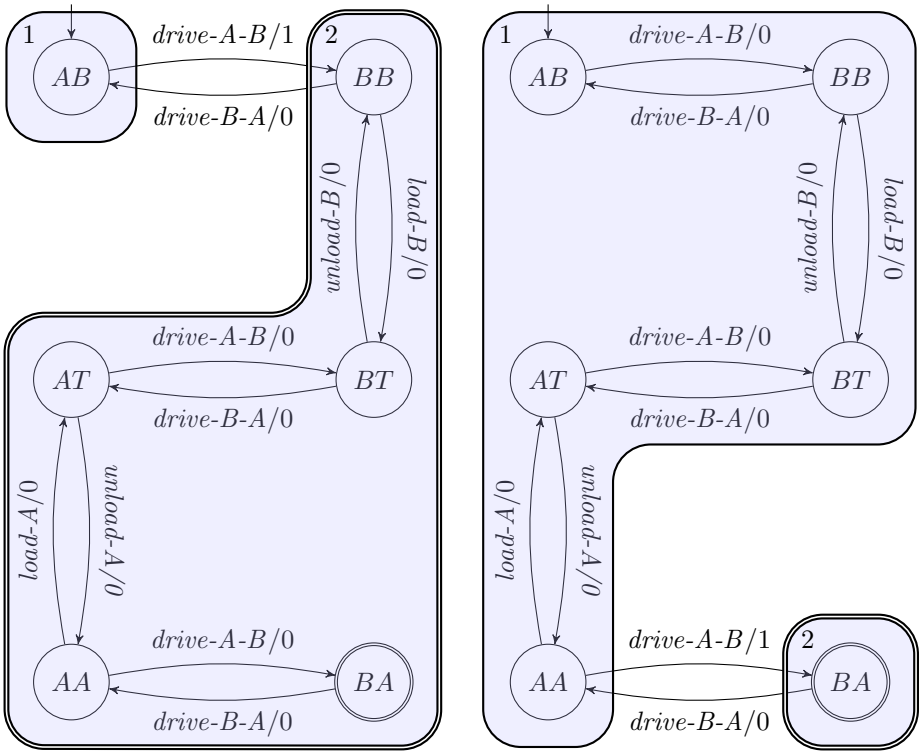


Figure 5.5: Transition cost partitioning over two abstractions of the example task.

erators. A context *is a propositional formula over atoms of* Π. Context-splitting o *with a context* φ *means replacing* o *with two new operators* o_φ *and* $o_{\neg\varphi}$. *The new operators have the same cost and the same effect as* o. *Their preconditions are* $pre(o_\varphi) = pre(o) \wedge \varphi$ *and* $pre(o_{\neg\varphi}) = pre(o) \wedge \neg\varphi$.

Context splitting is a task transformation that does not affect the optimal goal distance of any state:

Proposition 5.1. *Let* Π *be an ADL planning task with operators* $\mathcal{O}$. *For operator* $o \in \mathcal{O}$ *and context* φ, *let* o_φ *and* $o_{\neg\varphi}$ *be the two new operators resulting from context-splitting* o *with* φ. *Let* Π' *denote the task that only differs from* Π *in its operator set* $\mathcal{O}' = (\mathcal{O} \setminus \{o\}) \cup \{o_\varphi, o_{\neg\varphi}\}$.

For all states s *of* Π *(and* Π'*) it holds that* $h^*_\Pi(s) = h^*_{\Pi'}(s)$.

Proof: We can associate every s-plan π in Π with an s-plan π' in Π' of the same cost and vice versa.

From π' to π, we simply replace every occurrence of an operator o_φ or $o_{\neg\varphi}$ with the original operator o. This is possible because these operators only differ in the precondition and $pre(o_\varphi) \models pre(o)$ and $pre(o_{\neg\varphi}) \models pre(o)$.

From π to π' we check for every occurrence of o if φ is true in the state s' in which operator o is applied. If yes, we replace o with o_φ, otherwise we replace it with $o_{\neg\varphi}$.

These operators are applicable and have the same effect and the same cost as the original operator o. □

The theorem ensures that an admissible heuristic estimate for the transformed task is also an admissible estimate for the original task.

For the purpose of this thesis, we only consider a special kind of context splitting that is applicable to SAS$^+$ planning tasks where operator preconditions are partial states instead of arbitrary formulas over atoms. Instead of considering one context and distinguishing states where this context holds from those where it does not, we consider a set of mutually exclusive and jointly exhaustive contexts.

Definition 5.4 (context splitting for SAS$^+$)**.** *Let Π be a planning task and o one of its operators. A set of partial states C is a* set of contexts *for o if for every state s in which o is applicable there is exactly one $c \in C$ such that $s \models c$.* Context-splitting *o with a set of contexts $C = \{c_1, \ldots, c_n\}$ means replacing o with n new operators $o_{c_1}, \ldots, o_{c_n}$. The new operators have the same cost and the same effect as o. Their preconditions are $pre(o_{c_i}) = pre(o) \cup c_i$.*

When treating partial states as conjunctions of atoms, it is easy to see that context-splitting an operator o with a set of contexts $C = \{c_1, \ldots, c_n\}$ is the same as first splitting o with context c_1 and then successively splitting $o_{\neg c_i}$ with c_{i+1} for $1 \leq i < n$. Since contexts in C are mutually exclusive, terms of $\neg c_i \wedge c_j$ are equivalent to c_j. Since they are jointly exhaustive, there is no state in which $o_{\neg c_n}$ with the precondition $pre(o) \wedge \bigwedge_{i=1}^{n} \neg c_i$ is applicable. Proposition 5.1 thus applies to context-splitting over sets of contexts.

In the extreme case, each operator o of task Π is split with a set of contexts that contains one context for each state in which o is applicable. This results in a "disambiguated" task Π^{D} with a new operator for each transition that was previously induced by o. Operator cost partitioning over Π^{D} can thus distribute the cost of each transition of Π independently.

The same idea can also be expressed with *state-dependent operator costs* (Ivankovic et al., 2014). A state-dependent cost function or *transition cost function* assigns a cost to each transition instead of each operator. All concepts involving operator cost functions can be extended to transition cost functions in a straight-forward way. For example, the cost of an s-plan $\pi = \langle o_1, \ldots, o_n \rangle$ under transition cost function $cost'$ is $\sum_{i=1}^{n} cost'(s_{i-1} \xrightarrow{o_i} s_i)$, where $s = s_0, \ldots, s_n = s[\![\pi]\!]$ are the states visited by the plan π. In the same way, operator cost partitioning can be extended to *transition cost partitioning* (Keller et al., 2016).

Definition 5.5 (non-negative transition cost partitioning)**.** *Let Π be a planning task with transitions $\mathcal{T}$ and cost function cost. A* non-negative transition cost partitioning *for Π is a tuple $\langle cost_1, \ldots, cost_n \rangle$ where $cost_i : \mathcal{T} \to \mathbb{R}_0^+$ for $1 \leq i \leq n$ and*

$$\sum_{i=1}^{n} cost_i(s \xrightarrow{o} s') \leq cost(o) \text{ for all } s \xrightarrow{o} s' \in \mathcal{T}.$$

It is easy to see that transition cost partitioning makes admissible heuristics additive by interpreting transition cost functions as operator cost functions of the disambiguated task Π^{D}. An alternative direct proof of this is shown by Keller et al. (2016).

Proposition 5.2. *Let Π be a planning task, let $h_1, \ldots, h_n$ be admissible heuristics for Π, and let $P = \langle cost_1, \ldots, cost_n \rangle$ be a non-negative transition cost partitioning for Π. Then $h_P(h_1, \ldots, h_n, s) = \sum_{i=1}^{n} h_i(s, cost_i)$ is an admissible heuristic estimate for s.*

Analogously to $h^{\text{OCP+}}$ and h^{OCP}, we define an optimal transition cost partitioning and the corresponding heuristic.

Definition 5.6 (optimal non-negative transition cost partitioning)**.** *Let Π be a planning task with transitions $\mathcal{T}$ and let $h_1, \ldots, h_n$ be admissible heuristics for Π. Let $\mathcal{P}_{\mathcal{T}}^{+}$ be the set of all non-negative transition cost partitionings. A partitioning $P^* \in \mathcal{P}_{\mathcal{T}}^{+}$ is* optimal *in state s if*

$$P^* \in \arg\max_{P \in \mathcal{P}_{\mathcal{T}}^{+}} h_P(h_1, \ldots, h_n, s).$$

The optimal non-negative transition cost partitioning heuristic $h^{\text{TCP+}}$ *for $h_1, \ldots, h_n$ is the function*

$$h^{\text{TCP+}}_{\{h_i | 1 \leq i \leq n\}}(s) = \max_{P \in \mathcal{P}_{\mathcal{T}}^{+}} h_P(h_1, \ldots, h_n, s).$$

Revisiting the example in Figure 5.5, we can now see that the transition cost partitioning shown next to the operators has a heuristic value of 1 in both abstractions. As in the figure, we use XY to represent the state $\{\textit{pos-T} \mapsto X, \textit{pos-P} \mapsto Y\}$. The transition cost partitioning assigns 1 to $AB \xrightarrow{\textit{drive-A-B}} BB$ in the first abstraction and to $AA \xrightarrow{\textit{drive-A-B}} BA$ in the second abstraction, while all other transitions have a cost of 0.

Every operator cost function *cost* can be seen as a transition cost function *cost′* with $\textit{cost}'(s \xrightarrow{o} s') = \textit{cost}(o)$, so each operator cost partitioning is a transition cost partitioning. Therefore, $h^{\text{TCP+}}$ dominates $h^{\text{OCP+}}$. The example in Figure 5.5 shows that this dominance can be strict.

5.3. General Transition Cost Partitioning

We have seen that non-negative operator cost partitioning can be generalized in two different ways. Allowing cost functions to take negative values makes it possible to raise operator costs above their original costs in abstractions where they matter most. By partitioning transition costs instead of operator costs, we can distribute the costs differently in different contexts in which an operator can be applied. These two generalizations are orthogonal to each other and are easy to combine.

Definition 5.7 (general transition cost partitioning)**.** *Let* Π *be a planning task with transitions* $\mathcal{T}$ *and cost function cost. A* general transition cost partitioning *for* Π *is a tuple* $\langle cost_1, \ldots, cost_n \rangle$ *where* $cost_i : \mathcal{T} \to \mathbb{R}$ *for* $1 \leq i \leq n$ *and*

$$\sum_{i=1}^{n} cost_i(s \xrightarrow{o} s') \leq cost(o) \text{ for all } s \xrightarrow{o} s' \in \mathcal{T}.$$

As before, we can show that such partitionings make admissible heuristics additive:

Proposition 5.3. *Let* Π *be a planning task, let* $h_1, \ldots, h_n$ *be admissible heuristics for* Π*, and let* $P = \langle cost_1, \ldots, cost_n \rangle$ *be a general transition cost partitioning for* Π*. Then* $h_P(h_1, \ldots, h_n, s) = \sum_{i=1}^{n} h_i(s, cost_i)$ *is an admissible heuristic estimate for* s*.*

The proof is a direct extension of the proof for Theorem 5.1 to transition cost functions. We do not repeat it here as it adds nothing new, but it is contained in the paper by Keller et al. (2016). The heuristic h^{TCP} can now be defined in the obvious way:

Definition 5.8 (optimal general transition cost partitioning)**.** *Let* Π *be a planning task with transitions* $\mathcal{T}$ *and let* $h_1, \ldots, h_n$ *be admissible heuristics for* Π*. Let* $\mathcal{P}_\mathcal{T}$ *be the set of all general transition cost partitionings. A partitioning* $P^* \in \mathcal{P}_\mathcal{T}$ *is* optimal *in state* s *if*

$$P^* \in \arg\max_{P \in \mathcal{P}_\mathcal{T}} h_P(h_1, \ldots, h_n, s).$$

The optimal general transition cost partitioning heuristic h^{TCP} *for* $h_1, \ldots, h_n$ *is the function*

$$h^{\text{TCP}}_{\{h_i | 1 \leq i \leq n\}}(s) = \max_{P \in \mathcal{P}_\mathcal{T}} h_P(h_1, \ldots, h_n, s).$$

We already know that $h^{\text{OCP+}} \leq h^{\text{OCP}}$, that $h^{\text{OCP+}} \leq h^{\text{TCP+}}$, and that both dominance relations are strict. The examples in Figures 5.1 and 5.5 also show that h^{OCP} and $h^{\text{TCP+}}$ are incomparable. Every general operator cost function and every non-negative transition cost function is a general transition cost function, so $h^{\text{OCP}} \leq h^{\text{TCP}}$ and $h^{\text{TCP+}} \leq h^{\text{TCP}}$ hold as well. But are these dominance relations also strict?

Figure 5.6 shows an example of a general transition cost partitioning that yields a higher heuristic value than both h^{OCP} and $h^{\text{TCP+}}$, showing that the dominance is indeed strict in both cases. In the first abstraction, the initial state is also a goal state, so its heuristic value cannot exceed 0 under any cost function. The heuristic value of the second abstraction is determined by the cost of transition $BT \xrightarrow{\textit{drive-B-A}} AT$. Without assigning a negative cost to this transition in the first abstraction, its cost in the second abstraction cannot exceed 1, thus $h^{\text{TCP+}}(s_{\text{I}}) = 0 + 1$. The overall heuristic value can be increased by assigning a higher cost to this transition in the second abstraction, compensated by a negative cost in the first abstraction. However, since the operator *drive-B-A* induces a self-loop in the first abstraction, its cost should not be negative in all contexts. Otherwise, there would be a negative cost cycle which would make

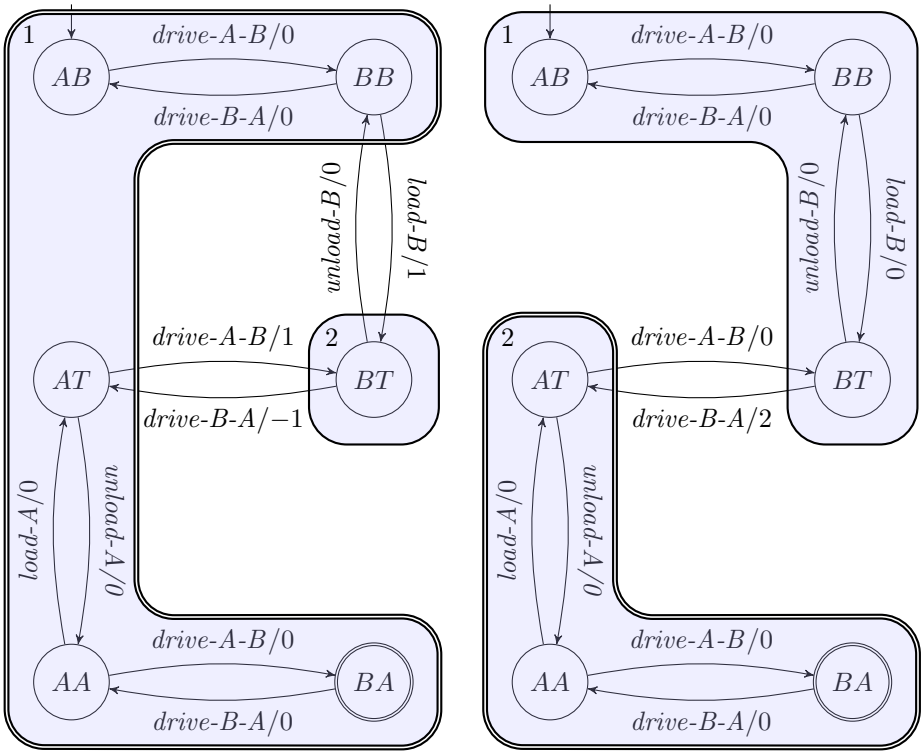


Figure 5.6: General transition cost partitioning over two abstractions of the example task. In the example $h^{\text{OCP}}(s_{\text{I}})$ and $h^{\text{TCP+}}(s_{\text{I}})$ are 1 but $h^{\text{TCP}}(s_{\text{I}})$ is 2.

the heuristic value of the first abstraction $-\infty$. Therefore, $h^{\text{OCP}}(s_{\text{I}}) = 0 + 1$. With the general transition cost partitioning shown in the figure, the cost of *drive-B-A* can be decreased only in the context of state BT in the first abstraction and raised in the second. The resulting heuristic value is $h^{\text{TCP}}(s_{\text{I}}) = 0 + 2$.

6. Experiments

The previous sections show that partitioning costs between abstraction heuristics can lead to higher heuristic values than their maximum. But does this actually happen in practice? To answer this question, we now compare different methods to combine a collection of projection heuristics. For now, we mostly focus on the heuristic quality of the combination, not on the time it takes to compute the heuristic value. Parts II and III introduce faster alternatives to certain cost partitioning methods.

We use projections for all patterns up to a given size k and call this collection All_k. For example, All_1 is the set of all atomic projections while All_2 additionally contains all projections to two variables. Since the number of projections quickly grows with k, we only consider values of k up to 3. Non-negative cost partitioning cannot benefit from patterns in All_k that are not interesting according to the definition of Pommerening, Röger, and Helmert (2013). We thus also consider the subset of interesting patterns in All_k, which we call Int_k.

6.1. Non-negative Operator Cost Partitioning

To measure the effectiveness of optimal cost partitioning as a heuristic combination method, we compare it to two other methods of combining the same abstraction heuristics: maximization and the canonical heuristic. Both of them can be seen as cost partitioning methods that are not guaranteed to be optimal. However, we compute both maximization and the canonical heuristic according to their original definitions which does not require involving an LP solver. Our implementation of optimal cost partitioning explicitly generates the abstract transition systems and the linear program introduced at the end of Chapter 4. Rather than re-creating the LP for every state, we keep the constraints in memory and only update the objective coefficients that change from state to state.

Figure 6.1 shows the heuristic values of the initial states from our benchmark suite. The first plot compares non-negative operator cost partitioning ($h^{\text{OCP+}}$) to maximization. Maximization rarely achieves a heuristic value above 20. This is not surprising because the heuristic value achievable by maximization is limited by the size of the largest projection and the largest operator cost. Even projections to three variables usually capture only a small aspect of the task, have few abstract states, and produce weak heuristic estimates. Combining many of these weak estimates instead of selecting the best one often leads to a much better heuristic estimate.

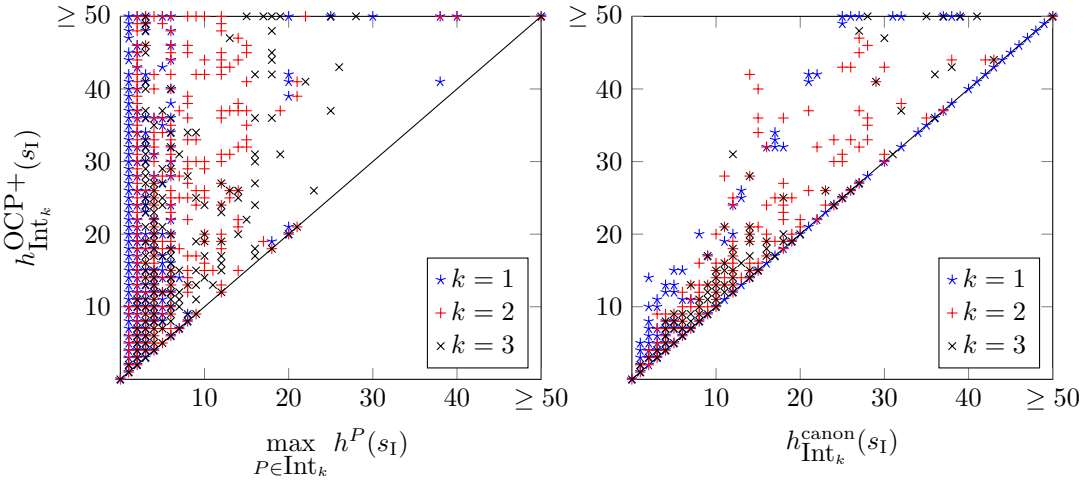


Figure 6.1: Heuristic values of the initial state. Heuristic combination of $h^{\text{OCP+}}$ on the y-axis is compared to combining the same projection heuristics with maximization (left) and the canonical heuristic (right).

As the canonical heuristic can be seen as a specific non-negative operator cost partitioning, the optimal operator cost partitioning dominates it. This can be seen in the second plot in Figure 6.1, where all marks are above the diagonal. Compared to maximization there are more cases where optimal cost partitioning does not increase the heuristic value, i.e. where the implicit cost partitioning found by the canonical heuristic is already optimal among the non-negative operator cost partitionings. However, there is also a significant number of tasks where optimal cost partitioning improves the heuristic. Some of the improvements are remarkable, for example in the domain FloorTile where the estimates are almost doubled. These instances fall on a line which clearly stands out in the plot. Even small improvements in heuristic value can lead to significant pruning, so large improvements like this demonstrate that high-quality estimates can be extracted from simple heuristics with operator cost partitioning. Can the extensions we proposed improve the heuristic even further?

6.2. General Operator Cost Partitioning

We first look at the extension to general operator costs, which just drops the non-negativity constraints in our implementation. Figure 6.2 shows how the initial heuristic values improve when allowing the use of general cost functions only on the "interesting" projections. Allowing negative costs can significantly increase the heuristic values even on the restricted sets of projections Int_k.

As we discussed in Section 5.1, the definition of "interesting" is too strict for general cost partitioning. Figure 6.3 thus shows how the heuristic values improve when all

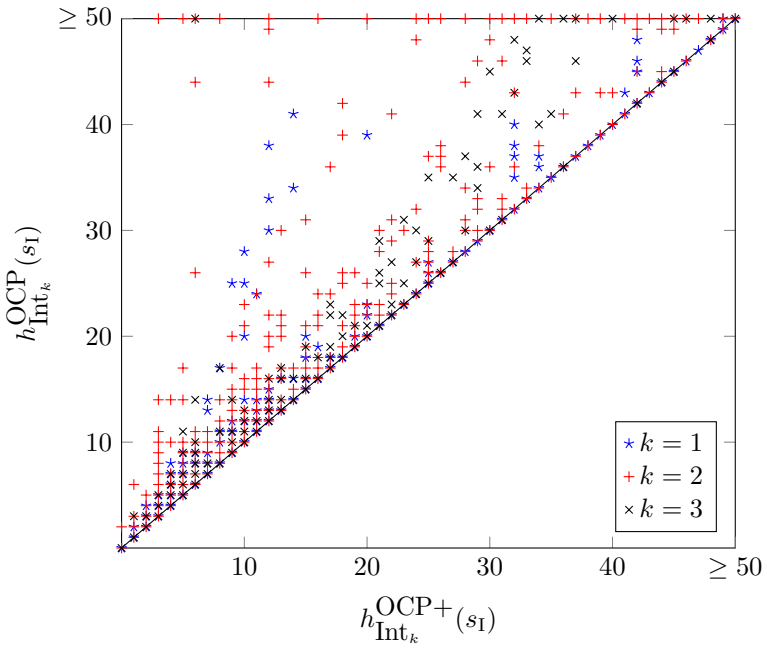


Figure 6.2: Heuristic values of the initial state. Heuristic combination with non-negative ($h^{\mathrm{OCP+}}$) and general (h^{OCP}) operator cost partitioning.

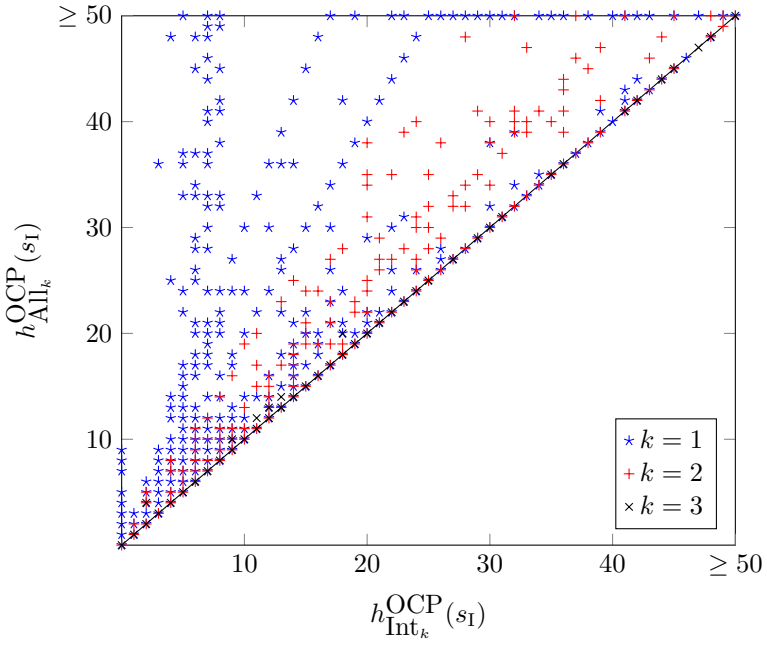


Figure 6.3: Heuristic values of the initial state. Heuristic combination with general operator cost partitioning (h^{OCP}) with and without projections that are not interesting for non-negative cost partitioning.

projections up to size k are used. Note that Figure 6.3 shows the improvement of $h^{\text{OCP}}_{\text{All}_k}$ over $h^{\text{OCP}}_{\text{Int}_k}$ not over $h^{\text{OCP+}}_{\text{Int}_k}$ as in Figure 6.2, so compared to $h^{\text{OCP+}}_{\text{Int}_k}$ the improvement is even higher. As we have seen before, instances from the same domain tend to behave similarly and fall on a line in the plot. In this case the most prominent lines are caused by the domains Freecell, Trucks, and Miconic.

We see that considering "uninteresting" patterns is critical for heuristic quality, especially when combining atomic projections. The gain from additional patterns is strongest for atomic projections ($k = 1$), weaker for projections to two variables ($k = 2$) and almost non-existent for larger patterns. This could be due to several reasons. When considering only goal variables (i.e. Int_1), other variables do not influence the heuristic value. Interesting patterns of larger size (i.e. Int_2 and Int_3) can contain non-goal variables. With increasing pattern size it becomes more likely that all variables are part of at least one pattern and can influence the heuristic value. Another factor could be that heuristic quality generally increases with the size of the considered pattern collection. Maybe most of the heuristic values of $h^{\text{OCP}}_{\text{Int}_3}$ are already perfect and we cannot expect an additional improvement when using more patterns?

To answer this question, we obtained optimal heuristic values for 1220 tasks in our benchmark set[1] by running several state-of-the-art planners with LM-cut (Helmert and Domshlak, 2009), iPDB (Haslum et al., 2007), merge-and-shrink (Helmert et al., 2014; Sievers, Wehrle, and Helmert, 2014), cost-partitioned Cartesian abstractions (Seipp, Keller, and Helmert, 2017b), potential heuristics (which we introduce in Part III) (Seipp, Pommerening, and Helmert, 2015), the state equation heuristic (Bonet, 2013), and symbolic search (Torralba, Linares López, and Borrajo, 2016).

With this data, we can evaluate how often a given heuristic is perfect on the initial state. However, not all methods finished their computation on all tasks. For example, computing all projections to three variables was not possible in 31 out of 50 tasks in the domain Airport. The canonical heuristic involves a step with exponential complexity in the number of projections so it also could not be computed on all tasks. And while computing an optimal operator cost partitioning for a set of abstractions is polynomial, it involves solving a linear program with one constraint for each abstract transition.

We thus have to restrict our evaluation to the subset of 182 tasks where the optimal heuristic value and the initial heuristic value for all methods could be computed within our resource limits of 30 minutes and 2 GB. Table 6.1 lists the number of times each heuristic was perfect on these 182 tasks. Better heuristic combination methods and larger sets of abstractions increase the number of tasks with a perfect heuristic estimate as expected. Nevertheless, there is still some room for improvement for all collection sizes. To answer our earlier question: the values of $h^{\text{OCP}}_{\text{Int}_3}$ are perfect in a lot of cases, but this does not completely explain why we see no additional benefit when considering more patterns with $h^{\text{OCP}}_{\text{All}_3}$.

[1] We submitted most of the resulting values as upper and lower bounds to the planning task repository `http://api.planning.domains/` (Muise, 2016) to make them publicly available.

			k		
			1	2	3
Int_k	Maximization	$\max_{P \in \text{Int}_k} h^P$	2	7	22
Int_k	Canonical heuristic	$h^{\text{canon}}_{\text{Int}_k}$	7	47	73
Int_k	Non-negative operator cost partitioning	$h^{\text{OCP+}}_{\text{Int}_k}$	37	73	127
Int_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{Int}_k}$	37	84	135
All_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{All}_k}$	43	123	136

Table 6.1: Number of times different heuristics produce perfect heuristic estimates in the initial state in 182 commonly solved tasks.

6.3. Non-negative Transition Cost Partitioning

Transition cost partitioning is an orthogonal extension to using general cost functions. So far, we have not discussed a method to compute an optimal transition cost partitioning efficiently. We therefore implemented an exponential version based on context splitting: we explicitly generate the full transition system of the task, then treat each transition as a separate operator and compute an optimal operator cost partitioning on the resulting task. Since this involves generating the full state space of the task as a first step, this method is certainly not useful as a heuristic in practice. If the transition system is small enough to be represented in memory, then Dijkstra's algorithm (Dijkstra, 1959) can be used to answer shortest path queries much faster than an A^* search which computes an optimal transition cost partitioning for each state. The inefficient computation is enough to get an idea of the heuristic quality, though. We circle back to transition cost partitioning in Part III and show methods to compute it efficiently in some cases.

The value of $h^{\text{TCP+}}(s_{\text{I}})$ was never higher than that of $h^{\text{OCP+}}(s_{\text{I}})$ for the tasks where we could compute $h^{\text{TCP+}}(s_{\text{I}})$. In these tasks an optimal non-negative operator cost partitioning also is optimal among the non-negative transition cost partitionings. This could be an effect of only considering small tasks and relatively small projections. (Remember that since we are restricting costs to non-negative numbers, projections that do not include a goal variable are uninteresting.)

Transition cost partitioning can only give higher heuristic values than operator cost partitioning if it makes sense to distinguish different contexts in which an operator can be applied. For example, this is the case if an operator is required more than once in an optimal plan and the different uses are recognized by different abstractions. In the introductory example of Section 5.2 (see Figure 5.5) *drive-A-B* was such an operator. Such complex interactions do not seem to occur in small projections of simple tasks but we do see evidence of it when we also consider general cost functions.

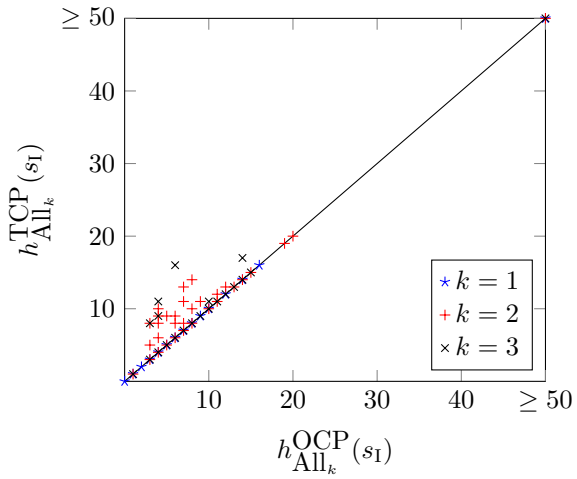


Figure 6.4: Heuristic values of the initial state. Heuristic combination with general operator (h^{OCP}) and transition (h^{TCP}) cost partitioning.

6.4. General Transition Cost Partitioning

Figure 6.4 shows that there is some benefit in transition cost partitioning with general cost functions. Heuristic values can exceed those of h^{OCP} in projections to at least two variables.

The resulting heuristic is highly accurate as we see by comparing it to optimal heuristic values again. Table 6.2 shows the number of times different heuristics produced optimal values. Compared to Table 6.1 the values are lower because we could only compute all heuristics in 87 tasks. But comparing the number within each table shows the same qualitative result: every row of the table shows at least as many task with optimal heuristic values as the preceding row, confirming that the heuristic in that row dominates the heuristics in the preceding rows. In most cases there are instances where only the dominating heuristic produces the optimal value, so this dominance is frequently strict. Using cost partitioning it is possible to combine many weak estimates into high quality heuristics. We get the highest heuristic quality with both general costs and context splitting. The resulting heuristic values for the collection All_3 are perfect in all tasks where we could compute them. But even for smaller patterns h^{TCP} often achieves perfect heuristic values.

6.5. Computation Time

So far we only considered the heuristic quality of cost partitioning. To be useful in practice, a heuristic also should be computable in a reasonable time. We thus estimate the expansion rate of the different approaches by measuring the number of expansions

			k		
			1	2	3
Int_k	Maximization	$\max_{P \in \text{Int}_k} h^P$	2	2	11
Int_k	Canonical heuristic	$h^{\text{canon}}_{\text{Int}_k}$	4	34	44
Int_k	Non-negative operator cost partitioning	$h^{\text{OCP+}}_{\text{Int}_k}$	34	37	72
Int_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{Int}_k}$	34	43	74
All_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{All}_k}$	35	67	74
All_k	General transition cost partitioning	$h^{\text{TCP}}_{\text{All}_k}$	35	75	87

Table 6.2: Number of times different heuristics produce perfect heuristic estimates in the initial state in 87 commonly solved tasks.

done in the first second after starting the search. This excludes the time for preprocessing, such as finding the set of maximally additive sets for the canonical heuristic or constructing the LP model for cost partitioning heuristics. If a method exceeded the resource limits during preprocessing, we counted this as an evaluation rate of 0 states per second. On the other hand, a good heuristic can solve many tasks in under a second. In such cases, we extrapolated the evaluation rate from the observed performance as the number of successful evaluations divided by the time spent in the search.

Figure 6.5 shows the estimated evaluation rates and the distribution over tasks. Each line shows the number of tasks with at most a certain evaluation rate for one of the heuristics. For example, the line for $h^{\text{canon}}_{\text{Int}_1}$ goes through the point $(10^5, 99)$ which means that a search with this heuristic evaluates less than 10^5 states per second in only 99 out of the 1667 tasks of our benchmark set. For $h^{\text{OCP}}_{\text{All}_1}$ this is true in 1577 tasks.

The evaluation rates of the operator cost partitioning heuristics show that allowing general cost functions makes the LPs harder to solve, but the impact is relatively small. The two LPs have the same amount of variables and constraints, and they only differ in their variable bounds. The effect is thus caused by an increased structural difficulty and not by a larger model. The difference is most visible with $k = 2$ and problems where the heuristics have a low evaluation rate. This effect is no longer visible for $k = 3$. A larger effect can be seen when comparing the heuristics for different pattern collections. Restricting the set to interesting patterns (i.e. using Int_k instead of All_k) significantly increases the evaluation rate. The effect is similar for both $h^{\text{OCP+}}$ and h^{OCP}.

Our naive implementation of transition cost partitioning requires to make the full state space of a task explicit. As this is only possible in a small fraction of the tasks, most tasks fail before the first evaluation or do not manage to evaluate the initial state within a second. In the cases where the model can be built, evaluation rates are rarely higher than a thousand states per second while this is common for all other tested heuristics with $k = 1$. Larger pattern collections ($k = 2$ and $k = 3$) decrease the computation

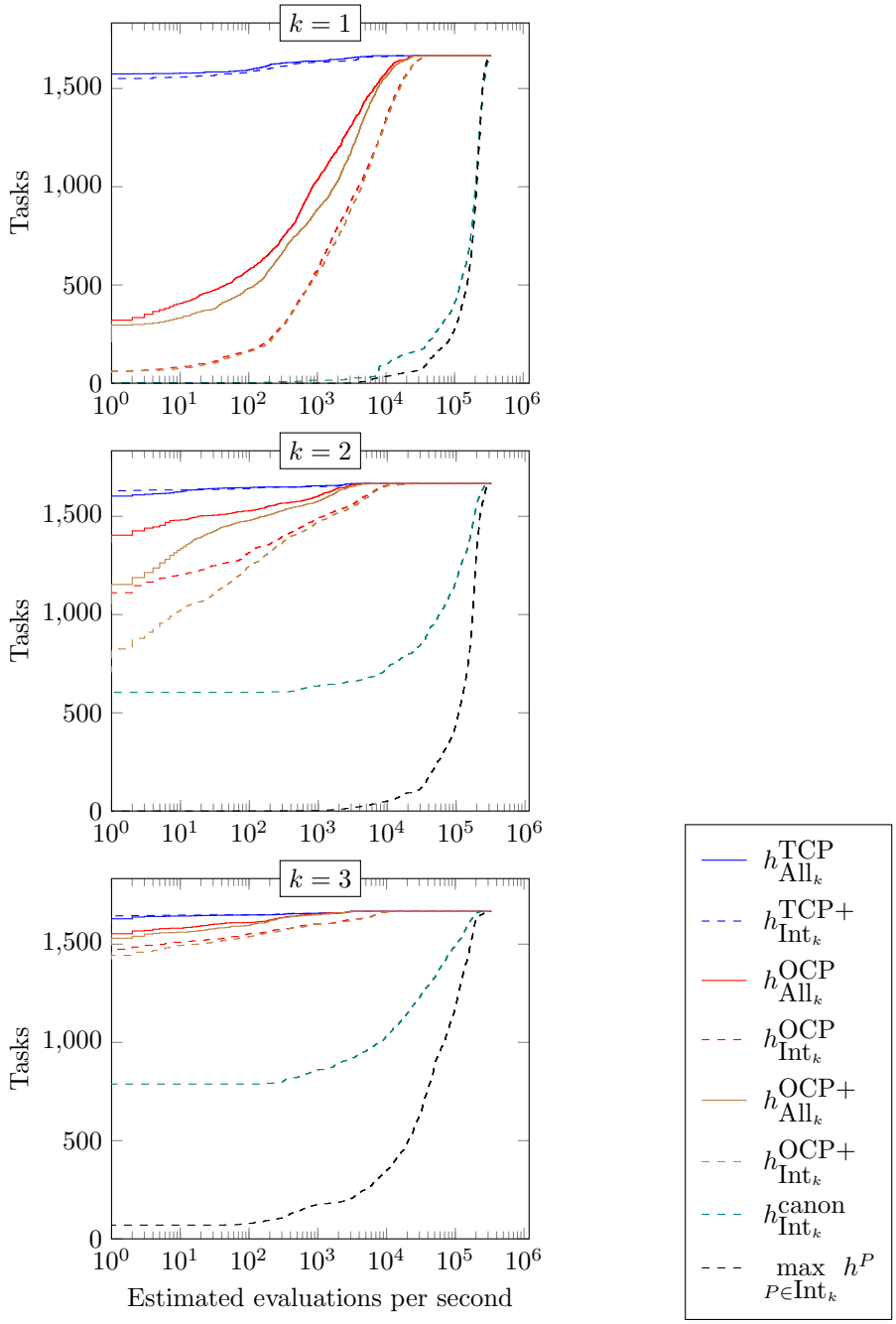


Figure 6.5: Evaluation rates estimated from the number of evaluations done in the first second after starting the search. If a search finishes or runs out of memory in $t < 1$ seconds after e evaluations, we estimate the evaluation rate as e/t. If the preprocessing did not finish, we count the evaluation rate as 0. All curves start at the origin and end in the upper right-hand corner. Faster methods bend stronger towards the lower right-hand corner.

			k		
			1	2	3
Int_k	Maximization	$\max_{P \in \text{Int}_k} h^P$	629	667	696
Int_k	Canonical heuristic	$h^{\text{canon}}_{\text{Int}_k}$	717	656	597
Int_k	Non-negative operator cost partitioning	$h^{\text{OCP+}}_{\text{Int}_k}$	621	438	237
All_k	Non-negative operator cost partitioning	$h^{\text{OCP+}}_{\text{All}_k}$	560	340	171
Int_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{Int}_k}$	622	475	248
All_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{All}_k}$	601	357	155
Int_k	Non-negative transition cost partitioning	$h^{\text{TCP+}}_{\text{Int}_k}$	120	76	26
All_k	General transition cost partitioning	$h^{\text{TCP}}_{\text{All}_k}$	115	87	74

Table 6.3: Number of solved tasks.

speed in general. The canonical heuristic suffers particularly from this because its preprocessing step is exponential in the size of the pattern collection.

We can evaluate the trade-off between evaluation time and heuristic quality with the number of tasks that can be solved within the resource limits. Table 6.3 shows an overview of the heuristics considered here. None of the cost partitioning approaches reaches the same performance as simple maximization.

We have seen that cost partitioning and its extensions can significantly improve heuristic quality but this comes at a high cost in computation time. In most cases extending the collection of projection heuristics (increasing k or using All_k instead of Int_k) did not improve the heuristic quality enough to offset the additional time spent on its computation. This is different for maximization where projections of size 3 had the best trade-off between computation time and heuristic quality. Allowing general cost functions in $h^{\text{OCP+}}$, on the other hand, pays off in almost all cases. For example, coverage increases by 37 tasks between $h^{\text{OCP+}}_{\text{Int}_2}$ and $h^{\text{OCP}}_{\text{Int}_2}$, and by 41 task between $h^{\text{OCP+}}_{\text{All}_1}$ and $h^{\text{OCP}}_{\text{All}_1}$. The only two exceptions are the collections Int_1 and All_3. In the former collection, the increased computation time almost perfectly balances the increase in heuristic quality. In the latter, we have seen that h^{OCP} can almost never draw additional information from the uninteresting patterns, so it has no advantage over $h^{\text{OCP+}}$.

If only non-negative costs are considered, using uninteresting patterns is useless and reduces coverage. For example, 621 tasks are solved with $h^{\text{OCP+}}_{\text{Int}_1}$ and this drops to 560 if all patterns of size 1 are used ($h^{\text{OCP+}}_{\text{All}_1}$). Allowing negative costs recovers some but not all of this loss: 601 tasks are solved with $h^{\text{OCP}}_{\text{All}_1}$. The LPs for general cost partitioning are harder to solve because more interactions with non-goal variables are possible. This decreases coverage by 1 in two domains. On the other hand, general cost partitioning results in fewer expansions before the last f-layer in 30 domains, which results in better coverage in 18 of them. Compared to $h^{\text{OCP+}}_{\text{Int}_1}$ using general operator cost partitioning and

projections to all variables does not pay off overall because the resulting LPs get too large. The picture is similar for $k = 2$ and $k = 3$. For transition cost partitioning, on the other hand, the advantages of higher heuristic values outweigh the disadvantages of longer computation time when comparing $h^{\text{TCP+}}_{\text{Int}_2}$ to $h^{\text{TCP}}_{\text{All}_2}$ and $h^{\text{TCP+}}_{\text{Int}_3}$ to $h^{\text{TCP}}_{\text{All}_3}$.

Using general operator cost partitioning on just the interesting patterns ($h^{\text{OCP}}_{\text{Int}_k}$) seems like a viable compromise which achieves higher heuristic values than $h^{\text{OCP+}}_{\text{Int}_k}$ at little additional cost. However, even the best $h^{\text{OCP+}}$ configuration ($h^{\text{OCP}}_{\text{Int}_1}$) solves fewer tasks than simple maximization. None of these heuristics can compete with the state of the art. For example, the LM-cut heuristic solves 880 tasks of this benchmark set in the same resource limits. In Parts II and III we discuss heuristics related to general cost partitioning that close this gap.

7. Summary

In this part, we introduced two orthogonal extensions to non-negative operator cost partitioning. Allowing negative costs and partitioning the cost of transitions independently both add more choices to the set of cost partitionings. Since optimal cost partitioning heuristics are based on the *best* choice from this set, allowing more choices results in significantly higher heuristic values. We have shown that h^{OCP} and $h^{\text{TCP+}}$ dominate $h^{\text{OCP+}}$ but are incomparable and that h^{TCP} dominates all three heuristics. Figure 7.1 summarizes their relationship.

There already is a tractable method to compute $h^{\text{OCP+}}$ for a set of explicit state abstraction heuristics. It computes a linear program that is easily adapted to compute h^{OCP} by dropping the constraint that costs have to be non-negative. Experiments show a significant increase in heuristic quality when general costs are allowed, with only little computational overhead. With general costs more inference is possible, for example sometimes detecting unsolvability even if all abstractions are solvable.

General cost partitioning also can use abstractions to non-goal variables to increase the heuristic value, which is impossible for non-negative cost partitioning. Pommerening, Röger, and Helmert (2013) define conditions under which a pattern provides no additional power to $h^{\text{OCP+}}$. Excluding such uninteresting patterns reduces the model size without influencing heuristic accuracy. For h^{OCP} no analogous definition of "interesting" exists yet. We have seen that adding uninteresting patterns increases heuristic quality but reduces coverage because of the high computation cost. Finding criteria under which a pattern is interesting for h^{OCP} is thus a promising direction for future research.

We focused on optimal cost partitioning here, but general cost partitioning is also beneficial for cost partitioning that is not guaranteed to be optimal. Seipp, Keller,

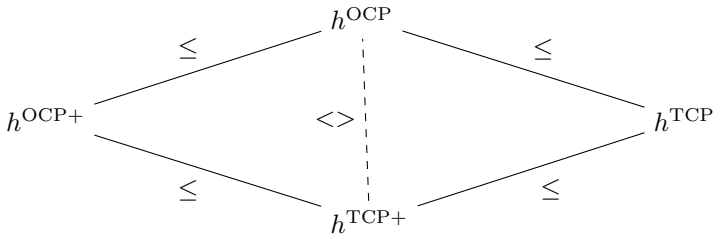


Figure 7.1: Dominance relations between different cost partitionings. The symbol <> is used to show that h^{OCP} and $h^{\text{TCP+}}$ are incomparable.

and Helmert (2017a) compare different operator cost partitioning algorithms and find that the best-performing algorithm is based on *saturated cost partitioning* (Seipp and Helmert, 2014). This algorithm processes heuristics in a fixed order and assigns each exactly the part of the cost that is necessary to maintain all heuristic values. The remaining costs are then used for the following heuristics. Using several diversified orders and allowing general cost functions turns this into a state-of-the-art heuristic which closely approximates optimal general operator cost partitioning (Seipp, Keller, and Helmert, 2017b).

While computing h^{OCP} is tractable for explicit state abstractions, this is less clear for $h^{\text{TCP+}}$ or h^{TCP}. The naive approach is to compute the optimal operator cost partitioning on a disambiguated task, i.e. using context splitting so every operator induces exactly one transition. The resulting LP contains one variable and one constraint for each combination of abstraction and transition; a number which is typically exponential in the size of the planning task. Keller et al. (2016) show how edge-valued decision diagrams (Ciardo and Siminiceanu, 2002) can be used to compute a saturated transition cost partitioning for Cartesian abstractions (Seipp and Helmert, 2013). While this is useful in practice, saturated cost partitioning is not guaranteed to be optimal and the size of the decision diagrams can become exponential. We return to the issue of computing h^{TCP} in Part III, where we introduce a tractable way to compute it for some abstractions.

Part II.

Operator Counting

8. Introduction to Operator Counting

While optimal cost partitioning heuristics produce good heuristic values, they are often too hard to compute to be useful in practice. However, existing heuristics have shown that it is feasible and beneficial to compute a linear program in every state of the search to obtain heuristic values (e.g. van den Briel et al., 2007; Karpas and Domshlak, 2009; Bonet, 2013; Pommerening, Röger, and Helmert, 2013). These heuristics are based on different sources of information: van den Briel et al. (2007) use network flows, Bonet (2013) uses the state equation of a Petri net, Karpas and Domshlak (2009) use landmarks and Pommerening, Röger, and Helmert (2013) use abstraction heuristics. In this part of the thesis, we introduce a framework that unifies these heuristics. This has two main advantages. Firstly, by bringing all heuristics into one common framework, it is possible to combine their information in a better way than maximization. Secondly, reasoning about heuristics, comparing and analyzing them, is easier if all involved heuristics "speak the same language". In our case this common language is *operator counting* which we introduce next. We then show how different heuristics can be expressed in that language and show some interesting theoretical properties. In particular, we show that operator counting has a strong connection to optimal cost partitioning.

The main idea of operator counting is to express properties that every plan has to satisfy as a set of constraints over the number of times an operator is used in the plan. Such constraints are called *operator-counting constraints*. Before we define them, we need one more auxiliary definition. For a given operator sequence $\pi = \langle o_1, \ldots, o_n \rangle$, we write the number of occurrences of operator $o \in \mathcal{O}$ in π as $\mathit{occur}_\pi(o)$, i.e. $\mathit{occur}_\pi(o) = |\{1 \leq i \leq n \mid o_i = o\}|$.

Definition 8.1 (operator-counting constraints)**.** *Let Π be a planning task with operators $\mathcal{O}$, and let s be one of its states. Let $\mathcal{Y}$ be a set of real-valued and integer variables, including a non-negative integer variable Count_o for each operator $o \in \mathcal{O}$ along with any number of additional variables. The variables Count_o are called* operator-counting variables*.*

A set of linear inequalities over $\mathcal{Y}$ is called an operator-counting constraint *for s if for every s-plan π, there exists a feasible solution f with $f(\mathsf{Count}_o) = \mathit{occur}_\pi(o)$ for all $o \in \mathcal{O}$.*

A constraint set *for s is a set of operator-counting constraints for s where the only common variables between constraints are the operator-counting variables.*

For example, consider the constraint $C_1 : \mathsf{Count}_{o_1} + \mathsf{Count}_{o_2} \geq 2$ which describes that at least two operators from every plan must be from $\{o_1, o_2\}$. The following inequalities

that use an additional variable X provide a slightly more complex example:

$$\mathsf{X} \geq \mathsf{Count}_{o_1} - \mathsf{Count}_{o_2} \tag{8.1}$$
$$\mathsf{X} \leq \mathsf{Count}_{o_2} \tag{8.2}$$

Together (8.1) and (8.2) form a constraint C_2 that expresses (in a complicated way) that $\mathsf{Count}_{o_1} \leq 2\mathsf{Count}_{o_2}$, i.e. that o_1 cannot be used more than twice as often as o_2. If this is true for every s-plan, the set of both inequalities is an operator-counting constraint. If both C_1 and C_2 are operator-counting constraints, then $\{C_1, C_2\}$ is a constraint set since X only occurs in C_2.

To explain the restriction on common variables in the definition of a constraint set, consider equations (8.1) and (8.2) individually. Constraint (8.1) by itself forms a trivial operator-counting constraint in every planning task because for every s-plan π it always has the solution f with $f(\mathsf{Count}_o) = \mathit{occur}_\pi(o)$ for all operators o and $f(\mathsf{X}) = f(\mathsf{Count}_{o_1}) - f(\mathsf{Count}_{o_2})$. Constraint (8.2) also is a trivial operator-counting constraint if it is considered individually: it always has a solution with $f(\mathsf{X}) = 0$. However their joint solutions are only those with $f(\mathsf{Count}_{o_1}) \leq 2f(\mathsf{Count}_{o_2})$ which is not true in every planning task. The set of both of these trivial operator-counting constraints is not a constraint set because the constraints share and interact through the variable X. The definition forbids this interaction of auxiliary variables with the same name, which could stand for unrelated quantities in different constraints.

Operator-counting constraints are necessary properties of plans, so an optimal plan satisfies all of them. We use this to define mathematical programs with optimal objective values that are admissible heuristic estimates:

Definition 8.2 (operator-counting integer/linear program)**.** *Let Π be a planning task with operators $\mathcal{O}$. The* operator-counting mixed integer program MIP_C *for constraint set C is:*

$$\textit{Minimize } \sum_{o \in \mathcal{O}} \mathit{cost}(o)\mathsf{Count}_o \textit{ subject to } C \textit{ and } \mathsf{Count} \geq \mathbf{0}.$$

The operator-counting linear program LP_C *is the LP-relaxation of* MIP_C.

Definition 8.3 (MIP and LP heuristic)**.** *Let Π be a planning task, and let $\mathcal{C}$ be a function that maps states s of Π to constraint sets for s. The* MIP heuristic $h^{\text{MIP}}_{\mathcal{C}}(s)$ *is the optimal objective value of the mixed integer program* $\text{MIP}_{\mathcal{C}(s)}$. *The* LP heuristic $h^{\text{LP}}_{\mathcal{C}}(s)$ *is the optimal objective value of the linear program* $\text{LP}_{\mathcal{C}(s)}$. *Infeasible MIPs/LPs are treated as having an optimal objective value of ∞.*

Theorem 8.1. *The heuristics $h^{\text{MIP}}_{\mathcal{C}}(s)$ and $h^{\text{LP}}_{\mathcal{C}}(s)$ are admissible.*

Proof: Let $\pi = \langle o_1, ..., o_n \rangle$ be an optimal s-plan. According to Definition 8.1 all constraints $C \in \mathcal{C}(s)$ have a solution f_C with $f_C(\mathsf{Count}_o) = \mathit{occur}_\pi(o)$ for all operators $o \in \mathcal{O}$. Since the variables Count_o are the only common variables of all C_i, the function

$f = \bigcup_{C \in \mathcal{C}(s)} f_C$ is well-defined, a solution of all $C \in \mathcal{C}(s)$ and therefore a solution of both MIP_C and LP_C. The value of solution f is

$$\sum_{o \in \mathcal{O}} \mathit{cost}(o) f(\mathsf{Count}_o) = \sum_{o \in \mathcal{O}} \mathit{cost}(o) \mathit{occur}_\pi(o) = \sum_{i=1}^{n} \mathit{cost}(o_i) = \mathit{cost}(\pi).$$

The optimal objective values of MIP_C and LP_C cannot exceed the value of any specific solution, so

$$h^{\text{LP}}_{\mathcal{C}}(s) \leq h^{\text{MIP}}_{\mathcal{C}}(s) \leq \mathit{cost}(\pi) = h^*(s).$$

□

The LP heuristic can be computed in time polynomial in the number and size of the constraints, while computing the MIP heuristic is NP-equivalent. However, in practice there are efficient MIP-solvers that can handle large problems and the MIP heuristic can lead to better heuristic values than the LP heuristic.

Proposition 8.1. *The heuristic $h^{\text{MIP}}_{\mathcal{C}}(s)$ strictly dominates $h^{\text{LP}}_{\mathcal{C}}(s)$.*

Proof: Every solution of a MIP is a solution of its LP-relaxation, so the optimal objective value of $\text{LP}_{\mathcal{C}(s)}$ is a lower bound for the optimal objective value of $\text{MIP}_{\mathcal{C}(s)}$.

A simple example shows that the dominance is strict: in a task modelling a two-digit binary counter (see Section 5.1) the only plan is $\langle o_1, o_2, o_1 \rangle$, so the constraints $\mathsf{Count}_{o_1} = 2\mathsf{Count}_{o_2}$ and $\mathsf{Count}_{o_1} + \mathsf{Count}_{o_2} \geq 1$ are both operator-counting constraints. The LP over these constraints has the solution $f_{\text{LP}}(\mathsf{Count}_{o_1}) = 2/3$, $f_{\text{LP}}(\mathsf{Count}_{o_2}) = 1/3$ which has an optimal objective value of 1 when all operators cost 1. The MIP does not permit this fractional solution and has an optimal objective value of 3, achieved with the solution $f_{\text{MIP}}(\mathsf{Count}_{o_1}) = 2$, $f_{\text{MIP}}(\mathsf{Count}_{o_2}) = 1$. □

If all operator costs of a planning task are integer, we can improve the LP heuristic estimate without losing admissibility by rounding up to the nearest integer. However, the previous example shows that the dominance between $h^{\text{MIP}}_{\mathcal{C}}$ and $h^{\text{LP}}_{\mathcal{C}}$ is strict even if the heuristic value is rounded up. It might be tempting to round up individual operator counts to get an integer solution, but this is not admissible and might even violate constraints. For example, the constraint $\mathsf{Count}_{o_1} = 2\mathsf{Count}_{o_2}$ is violated if the elements of f_{LP} are rounded up to 1.

Note that the requirement that an operator-counting constraint must have a feasible solution with $\mathsf{Count}_o = \mathit{occur}_\pi(o)$ for *every* plan π is stricter than necessary for admissibility. It is sufficient that whenever a solution exists, there is one optimal plan π^* such that all operator-counting constraints have a feasible solution f with $f(\mathsf{Count}_o) = \mathit{occur}_{\pi^*}(o)$. Similarly, we could allow operator-counting constraints to share some variables X if they guarantee that solutions are compatible, i.e. that for every choice of values for variables Count_o and X either all constraints have a solution with these values or none of them has one. We do not use such extensions here because

they complicate the definition and they are not necessary for the constraints we introduce in the following chapter. Sharing additional information between constraints is still an interesting idea which we discuss further in Chapter 12.

The compatibility of solutions is important as it guarantees that operator-counting constraints can easily be combined. Since adding constraints can only reduce the set of feasible solutions for an mixed integer/linear program, the resulting heuristic estimates can only increase with more constraints.

Proposition 8.2. *Let $\mathcal{C}, \mathcal{C}'$ be functions that map states s of a planning task Π to constraint sets for s such that $\mathcal{C}(s) \subseteq \mathcal{C}'(s)$ for all states s. Then the MIP/LP heuristic for $\mathcal{C}'$ dominates the respective heuristic for $\mathcal{C}$: $h^{\text{MIP}}_{\mathcal{C}} \leq h^{\text{MIP}}_{\mathcal{C}'}$ and $h^{\text{LP}}_{\mathcal{C}} \leq h^{\text{LP}}_{\mathcal{C}'}$.*

In particular, this means that combining operator-counting constraints into one constraint set dominates considering them individually and computing their maximum. We will later see examples where this dominance is strict.

Corollary 8.1. *Let Π be a planning task Π and s one of its states. Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be a constraint set for s. Then $h^{\text{MIP}}_{\mathcal{C}} \geq \max_{1 \leq i \leq n} h^{\text{MIP}}_{\{C_i\}}$ and $h^{\text{LP}}_{\mathcal{C}} \geq \max_{1 \leq i \leq n} h^{\text{LP}}_{\{C_i\}}$.*

9. Operator-Counting Constraints

We now discuss six types of operator-counting constraints that capture different state-of-the-art heuristics for optimal planning. The constraints cover three of the four previously mentioned heuristic classes: landmarks, delete relaxation, and abstraction. We will later discuss why critical path heuristics are missing from this list. Our last three examples are related to the state equation heuristic (van den Briel et al., 2007; Bonet, 2013) that seems not to fit into any of the four heuristic classes.

9.1. Action Landmarks

A *disjunctive action landmark* (Zhu and Givan, 2003; Helmert and Domshlak, 2009) for a state s is a set of operators of which at least one must be part of any s-plan. In our running example task from Section 2.1, the set $\{$*load-B*$\}$ is a simple landmark for s_{I} because every plan has to load the package at some point. If we duplicate all drive operators, so there is a fast and a slow way to drive along any road, then $\{$*drive-A-B*$_{slow}$, *drive-A-B*$_{fast}\}$ is also a landmark.

There are other types of landmarks called *fact landmarks* that are also often used, for example in the well-known LAMA (Richter and Westphal, 2010). Fact landmarks can be transformed to action landmarks, i.e. the set of all operators that add an atom from an unachieved fact landmark forms a disjunctive action landmark. For the rest of the thesis, we only consider disjunctive action landmarks and just refer to them as landmarks.

Using linear programming to derive heuristic estimates from landmarks was introduced by Karpas and Domshlak (2009) as cost partitioning for landmarks. Their LP assigns a value to each landmark and a (non-negative) cost to each combination of operator and landmark. The value of a landmark L then is the minimum cost of any $o \in L$ in the context of L. By the cost partitioning property, the cost of o in the context L summed over all L cannot exceed $\mathit{cost}(o)$. The LP formulation was improved by Keyder, Richter, and Helmert (2010). Bonet and Helmert (2010) introduced an alternative LP representation that directly fits into the operator-counting constraint framework and showed that it is the dual of the representation by Keyder et al.

Strengthening other heuristics with landmarks is not a new idea: Domshlak, Katz, and Lefler (2012) propose it for abstraction heuristics and Bonet (2013) for the LP-based state equation heuristic. He uses the same constraints as Bonet and Helmert (2010):

Definition 9.1 (landmark constraint)**.** *Let $L \subseteq \mathcal{O}$ be a disjunctive action landmark for state s of task Π. The* landmark constraint $c^{\mathrm{lm}}_{s,L}$ *for L is*

$$\sum_{o \in L} \mathsf{Count}_o \geq 1.$$

Since at least one operator of the landmark occurs in every s-plan, landmark constraints clearly meet the requirements of operator-counting constraints.

For an example of how such landmarks can be useful, consider our running example task under a cost function where driving between A and B has a cost of 3 while (un)loading the package costs 5. Assume we discovered that $\{\textit{drive-A-B}, \textit{load-B}\}$ and $\{\textit{drive-B-A}, \textit{load-B}\}$ are landmarks for the initial state. From each individual landmark we can determine that every plan must cost at least 3 (the cost of the cheapest operator in the landmark). The landmark constraint for the given landmarks are $\mathsf{Count}_{\textit{drive-A-B}} + \mathsf{Count}_{\textit{load-B}} \geq 1$ and $\mathsf{Count}_{\textit{drive-B-A}} + \mathsf{Count}_{\textit{load-B}} \geq 1$. Minimizing the total plan cost $3\mathsf{Count}_{\textit{drive-A-B}} + 3\mathsf{Count}_{\textit{drive-B-A}} + 5\mathsf{Count}_{\textit{load-B}}$ subject to these constraints establishes a better heuristic estimate of 5.

9.2. Delete Relaxation

The next type of operator-counting constraint is based on the *delete relaxation* (McDermott, 1999; Bonet and Geffner, 2001; Hoffmann and Nebel, 2001). Originally, delete relaxation was introduced for the STRIPS framework, where all planning variables are binary and (goal- and operator-) conditions are conjunctions of positive literals. In STRIPS a state can be represented as the set of variables that are true and operator effects either *add* or *delete* variables from this set. The delete relaxation of a task ignores all delete effects of operator. The resulting task is simpler and can be used to derive admissible heuristic estimates. The concept was later generalized to SAS$^+$ tasks (e.g. Domshlak, Hoffmann, and Katz, 2015).[1] In SAS$^+$ the task is not changed by delete relaxation, only its semantics are modified so variables accumulate their values.

Definition 9.2 (delete relaxation semantics)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, \mathit{cost} \rangle$ be a planning task. A partial state in the* delete relaxation *of Π is a set of atoms over $\mathcal{V}$. An operator $o \in \mathcal{O}$ is applicable in state s if $pre(o) \subseteq s$ and applying it results in the state $s[\![o]\!]^+ = s \cup \mathit{eff}(o)$. A state s is a goal state if $s_\star \subseteq s$. A* delete-relaxed s-plan *is an operator sequence that is successively applicable in s and ends in a goal state.*

Note that states of Π can be seen as states of the delete relaxation but not necessary the other way around because delete-relaxed states can contain more than one atom for a variable. Figure 9.1 shows the reachable state space of the delete relaxation of our

[1] The origin of this generalization is not entirely clear and is discussed by Domshlak, Hoffmann, and Katz.

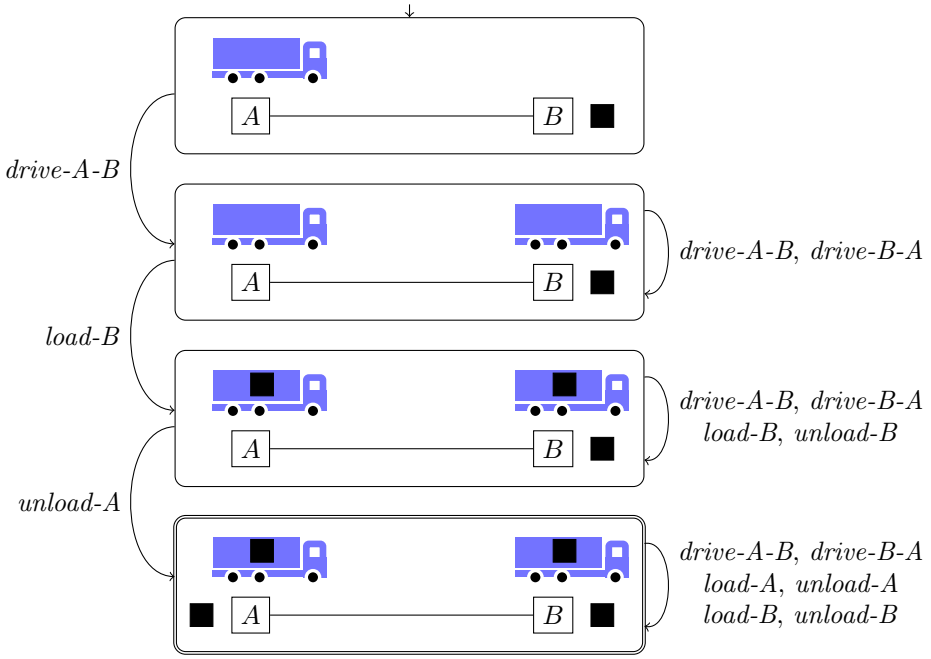


Figure 9.1: Reachable state space for the delete relaxation of our running example task.

example task. In the initial state, the only applicable operator is *drive-A-B* which adds the atom that the truck is at location B. In the delete relaxation this does not exclude that the truck is at location A at the same time. The resulting state contains both atoms $\langle \textit{pos-T}, A\rangle$ and $\langle \textit{pos-T}, B\rangle$. Driving between A and B again does not add additional atoms to the state, but loading the package adds $\langle \textit{pos-P}, T\rangle$ and unloading it afterwards adds $\langle \textit{pos-P}, A\rangle$.

Since $s[\![o]\!] \subseteq s[\![o]\!]^+$, an easy proof by induction shows that every s-plan also is a delete-relaxed s-plan (this can also be verified in Figure 9.1). Therefore, the cost of an optimal delete-relaxed plan cannot exceed the cost of an optimal plan and is an admissible heuristic, called the *delete-relaxation heuristic* h^+.

After an operator o is applied in the delete relaxation, applying it again does not add additional atoms to the state, so every occurrence of o except the first can be removed from a delete-relaxed s-plan. Because of $s \subseteq s[\![o]\!]$, applying operators cannot make an applicable operator inapplicable. Optimal delete-relaxed plans can thus be represented as a subset of operators. A valid order for such a set can be constructed greedily by repeatedly choosing, applying and removing an applicable operator from the set. Computing h^+ is thus in NP. In fact, the problem is NP-equivalent (Bylander, 1997) and can not even be approximated within a constant factor in polynomial time unless P $=$ NP (Betz and Helmert, 2009).

Imai and Fukunaga (2014, 2015) show that h^+ can be computed as the optimal objective value of a MIP that fits the operator-counting framework. The model uses the fol-

lowing integer variables to express necessary and sufficient properties of delete-relaxed s-plans π in a planning task with operators $\mathcal{O}$ and atoms $\mathcal{A}$:[2]

- $\mathsf{Used}_o \in \{0, 1\}$ for every $o \in \mathcal{O}$
  indicating whether o is part of π
- $\mathsf{Reached}_a \in \{0, 1\}$ for every $a \in \mathcal{A}$
  indicating whether a is reached by π
- $\mathsf{Achiever}_{o,a} \in \{0, 1\}$ for every $o \in \mathcal{O}$ and every $a \in \mathcal{A}$
  indicating whether o is the first operator in π that adds a
- $\mathsf{Time}_o \in \{0, \ldots, |\mathcal{O}|\}$ for every $o \in \mathcal{O}$
  indicating the time operator o is applied in the plan. The first operator can be applied at time point 0 and value $|\mathcal{O}|$ indicates that the operator is not used.
- $\mathsf{Time}_a \in \{0, \ldots, |\mathcal{O}|\}$ for every $a \in \mathcal{A}$
  indicating the time atom a is true for the first time. Value 0 means that a is already true in the state s.

Using these variables, Imai and Fukunaga define the following constraints:

$$\mathsf{Reached}_a = 1 \qquad \text{for all } a \in s_\star \tag{9.1}$$

$$[a \in s] + \sum_{\substack{o \in \mathcal{O} \\ a \in \mathit{eff}(o)}} \mathsf{Achiever}_{o,a} \geq \mathsf{Reached}_a \qquad \text{for all } a \in \mathcal{A} \tag{9.2}$$

$$\mathsf{Used}_o \geq \mathsf{Achiever}_{o,a} \qquad \text{for all } o \in \mathcal{O}, a \in \mathit{eff}(o) \tag{9.3}$$

$$\mathsf{Reached}_a \geq \mathsf{Used}_o \qquad \text{for all } o \in \mathcal{O}, a \in \mathit{pre}(o) \tag{9.4}$$

$$\mathsf{Time}_a \leq \mathsf{Time}_o \qquad \text{for all } o \in \mathcal{O}, a \in \mathit{pre}(o) \tag{9.5}$$

$$\mathsf{Time}_o + 1 \leq \mathsf{Time}_a + M(1 - \mathsf{Achiever}_{o,a}) \qquad \text{for all } o \in \mathcal{O}, a \in \mathit{eff}(o) \tag{9.6}$$

$$\mathsf{Reached}_a, \mathsf{Used}_o, \mathsf{Achiever}_{o,a} \in \{0, 1\} \qquad \text{for all } o \in \mathcal{O}, a \in \mathcal{A} \tag{9.7}$$

$$\mathsf{Time}_a, \mathsf{Time}_o \in \{0, \ldots, |O|\} \qquad \text{for all } o \in \mathcal{O}, a \in \mathcal{A} \tag{9.8}$$

A delete-relaxed s-plan must reach every goal atom (9.1). An atom can only be reached if it is true in s or there is an operator that adds the atom for the first time (9.2). An operator can only be a first achiever for an atom if it is used (9.3). An operator can only be used if all its preconditions are reached (9.4) before it is applied (9.5) Finally, if an operator is the first achiever of an atom then this atom must be added for the first time directly after the operator is used (9.6). In the last inequality M is a constant that is large enough to satisfy the inequality if $\mathsf{Achiever}_{o,a} = 0$. Since the time steps are limited to the number of operators, $M = |\mathcal{O}| + 1$ is sufficient.

[2] We use different variable names than Imai and Fukunaga.

Theorem 9.1 (Imai and Fukunaga, 2015)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, cost \rangle$ be a planning task with atoms $\mathcal{A}$. The optimal objective value of the MIP*

$$\textit{Minimize} \sum_{o \in \mathcal{O}} cost(o) \mathsf{Used}_o \textit{ subject to } (9.1)\text{–}(9.8)$$

is equal to the delete-relaxation heuristic $h^+(s)$.

The proof is based on the idea that every delete-relaxed plan π that mentions each operator at most once corresponds to a solution f of the MIP with $f(\mathsf{Used}_o) = occur_\pi(o)$ and vice versa. We refer to the original proof of Imai and Fukunaga for details.

To extend these constraints to an operator-counting constraint, we only have to link the variables Used_o to the operator-counting variables as follows.

Definition 9.3 (delete relaxation constraint)**.** *Let Π be a planning task with operators $\mathcal{O}$. The* delete-relaxation constraint c_s^+ *consists of the constraints* (9.1)–(9.8) *for Π and the constraint*

$$\mathsf{Count}_o \geq \mathsf{Used}_o \qquad \textit{for all } o \in \mathcal{O}. \tag{9.9}$$

To show that c_s^+ is indeed an operator-counting constraint, consider any s-plan π and remove all except the first occurrence of each operator. The result is a delete-relaxed plan π' for which a solution to constraints (9.1)–(9.8) exists. The additional constraint (9.9) is satisfied because $occur_\pi(o) \geq occur_{\pi'}(o)$ for all $o \in \mathcal{O}$. The following proposition then follows with Theorem 9.1.

Proposition 9.1. *Let $\mathcal{C}$ be the function that maps every state s to $\{c_s^+\}$. Then $h^{\mathrm{MIP}}_{\mathcal{C}} = h^+$.*

Imai and Fukunaga also introduce several variations of the basic model, and most of them can be expressed in the operator-counting framework. Firstly, they suggest solving the LP-relaxation instead of the MIP. This corresponds to using $h^{\mathrm{LP}}_{\mathcal{C}}$ instead of $h^{\mathrm{MIP}}_{\mathcal{C}}$. Secondly, they relax the notion of time by removing the variables Time_o, Time_a and constraints (9.5), (9.6), and (9.8). Removing constraints cannot reduce the set of solutions, so the resulting set of inequalities is still an operator-counting constraint. Thirdly, they suggest variable elimination techniques that reduce the size of the model without reducing the set of solutions. For example, if $\{o\}$ is a landmark, then Used_o can be replaced by 1 in all constraints. Finally, two extensions based on a relevance analysis and on inverse operators can reduce the set of feasible solutions, but not the set of solutions projected to the operator-counting variables. All such techniques can be used in the context of operator counting as well.

However two of their extensions, *immediate operator application* and *dominated actions*, are specific to the delete relaxation.

The first of these extensions forces the application of all 0-cost operators that are applicable in the initial state. This is fine in the delete relaxation because the application cannot increase the total cost or prevent any other operator application. In the context

of operator counting, this is not allowed (i.e. the constraint with this extension is no longer an operator-counting constraint). Consider our running example task with an added 0-cost operator that destroys the package. In the delete relaxation, applying this operator does not prevent achieving the goal afterwards, but there is no plan in the original problem that uses this operator.

Dominated actions can be eliminated from the model, but it depends on the definition of dominance whether this preserves the operator-counting constraint property. In the delete relaxation it is possible to ignore delete effects. Imai and Fukunaga's definition uses this and is therefore only valid in the delete relaxation. They consider the combination of the delete-relaxation constraint with net change constraints, which we introduce in Section 9.4. For this combination, they offer an alternative definition that considers delete effects for variables that occur in the operator's precondition. This makes excluding dominated actions in the delete-relaxation constraint compatible with net change constraints but is still not sufficient for general operator-counting constraints.

9.3. Post-hoc Optimization

Our question in Chapter 1 was how to derive admissible and informative heuristics from multiple admissible heuristics. We have already seen three methods to combine abstraction heuristics: using the maximum of all component heuristics is cheap to compute but can lead to weak heuristic estimates. The canonical heuristic dominates the maximum but uses a preprocessing step that can take time exponential in the number of component heuristics. Optimal cost partitioning dominates the canonical heuristic and can be computed in polynomial time. In practice, it yields high quality heuristic values but requires solving huge linear programs for every heuristic computation – up to millions of variables and billions of constraints for realistic problem sizes.

We now introduce operator-counting constraints that offer a new way to combine heuristic estimates. *Post-hoc optimization* is a middle-ground between the canonical heuristic and optimal cost partitioning and can be computed in polynomial time.

We first focus on PDB heuristics and generalize the results afterwards. Since PDB heuristics are admissible, we know that the PDB heuristic value cannot exceed the total cost incurred by all operators:

$$h^P(s) \leq \sum_{o \in \mathcal{O}} \mathit{cost}(o) \mathsf{Count}_o.$$

Operators which do not affect any variable in a pattern cannot contribute to the estimate of the PDB since they do not induce state changes in the abstract task. Therefore we can tighten the constraint for h^P by omitting variables for such "irrelevant" operators:

$$h^P(s) \leq \sum_{\substack{o \in \mathcal{O} \\ o \text{ affects } P}} \mathit{cost}(o) \mathsf{Count}_o. \tag{9.10}$$

This approach is not limited to PDB heuristics. The only properties of PDB heuristics that we exploited are that they are admissible and that certain operators do not contribute to the heuristic value of certain PDBs. This concept can be generalized using notation from cost partitioning. An operator o is irrelevant for a heuristic function h if the estimates of h are also admissible in a task where $\mathit{cost}(o)$ is set to 0.

Definition 9.4 (post-hoc optimization constraint)**.** *Let Π be a planning task with the cost function cost and let s be one of its states. Let Π' be like Π but with a non-negative cost function $\mathit{cost}' \leq \mathit{cost}$. If $h(s, \mathit{cost})$ is an admissible heuristic estimate for state s in Π', then the* post-hoc optimization constraint $c^{\text{pho}}_{s,h,\mathit{cost}'}$ *is*

$$\sum_{\substack{o \in \mathcal{O} \\ \mathit{cost}'(o) > 0}} \mathit{cost}(o)\mathsf{Count}_o \geq h(s, \mathit{cost})$$

To prove that $c^{\text{pho}}_{s,h,\mathit{cost}'}$ is an operator-counting constraint, consider any s-plan π. We use $\mathit{cost} \geq \mathit{cost}'$ and the admissibility of h to show that the assignment $f(\mathsf{Count}_o) = \mathit{occur}_\pi(o)$ satisfies the constraint:

$$\begin{aligned}
\sum_{\substack{o \in \mathcal{O} \\ \mathit{cost}'(o) > 0}} \mathit{cost}(o)\mathit{occur}_\pi(o) &\geq \sum_{\substack{o \in \mathcal{O} \\ \mathit{cost}'(o) > 0}} \mathit{cost}'(o)\mathit{occur}_\pi(o) \\
&= \sum_{o \in \mathcal{O}} \mathit{cost}'(o)\mathit{occur}_\pi(o) \\
&= \mathit{cost}'(\pi) \\
&\geq h(s, \mathit{cost})
\end{aligned}$$

The constraint (9.10) is a special case of a post-hoc optimization constraint. Due to the importance of PDB heuristics, we give a separate definition for them:

Definition 9.5 (PDB constraint)**.** *Let P be a pattern for a planning task with cost function cost. Further, let cost_P be the cost function with $\mathit{cost}_P(o) = 0$ for all operators o that do not affect P, and $\mathit{cost}_P(o) = \mathit{cost}(o)$ for all other operators. The* PDB constraint $c^{\text{PDB}}_{s,P}$ *is the post-hoc optimization constraint $c^{\text{pho}}_{s,h^P,\mathit{cost}_P}$.*

The post-hoc optimization heuristic *for a pattern collection $\mathcal{C}$ is $h^{\text{pho}}_{\mathcal{C}} = h^{\text{LP}}_{\{c^{\text{PDB}}_{s,P} \mid P \in \mathcal{C}\}}$.*

Operator-counting heuristics with PDB constraints offer a new way of combining heuristic estimates of a set of PDB heuristics. To analyze the connection to the canonical heuristic and optimal cost partitioning, we consider the dual of the operator-counting LP with PDB constraints for a pattern collection $\mathcal{C}$:

$$\text{Minimize} \sum_{o \in \mathcal{O}} \mathit{cost}(o)\mathsf{Count}_o \text{ subject to}$$

$$\sum_{\substack{o \in \mathcal{O} \\ o \text{ affects } P}} \mathit{cost}(o)\mathsf{Count}_o \geq h^P(s) \qquad \text{for all } P \in \mathcal{C} \tag{9.11}$$

$$\mathsf{Count}_o \geq 0 \qquad \text{for all } o \in \mathcal{O} \tag{9.12}$$

The dual has a variable Y_P for each constraint of type (9.11) and one constraint for each operator:

$$\text{Maximize} \sum_{P \in \mathcal{C}} \mathsf{Y}_P h^P(s) \text{ subject to}$$

$$\sum_{\substack{P \in \mathcal{C} \\ o \text{ affects } P}} \mathit{cost}(o)\mathsf{Y}_P \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O} \tag{9.13}$$

$$\mathsf{Y}_P \geq 0 \qquad \text{for all } P \in \mathcal{C} \tag{9.14}$$

A solution f of this LP is a non-negative cost partitioning into cost functions cost_P for each pattern P where $\mathit{cost}_P(o) = 0$ if o does not affect P and $\mathit{cost}_P(o) = \mathit{cost}(o)f(\mathsf{Y}_P)$ otherwise. That is, the costs of all relevant operators are scaled down by a factor that only depends on the pattern. The constraints (9.13) guarantee that the cost partitioning property is satisfied. Since this is a non-negative cost partitioning the resulting heuristic is dominated by $h^{\text{OCP+}}$.

On the other hand, it is easy to see that the heuristic dominates the canonical heuristic for the same pattern collection: let $A^{\max}$ be the additive set for which the canonical heuristic achieves its maximum and let f be the assignment with $f(\mathsf{Y}_P) = [P \in A^{\max}]$. Since the patterns in $A^{\max}$ are additive, f satisfies all constraints and the value of f is exactly $h^{\text{canon}}_{\mathcal{C}} = \sum_{P \in A^{\max}} h^P$. The optimal objective value of the LP is at least as high, so the post-hoc optimization heuristic dominates the canonical heuristic. If the variables Y_P are restricted to be binary variables, the dual LP exactly describes the canonical heuristic. An optimal solution f then selects a set of heuristics (those with $f(\mathsf{Y}_i) = 1$) and adds their heuristic values. The cost partitioning constraint ensures additivity in this case.

The following LP shows the differences between maximization, the canonical heuristic, post-hoc optimization and cost partitioning:

$$\text{Maximize} \sum_{P \in \mathcal{C}} h^P(s, \mathbf{Cost}^P) \text{ subject to}$$

$$\sum_{P \in \mathcal{C}} \mathsf{Cost}^P_o \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O} \tag{9.15}$$

$$\mathsf{Cost}^P_o \geq 0 \qquad \text{for all } o \in \mathcal{O} \text{ and } P \in \mathcal{C} \tag{9.16}$$

$$\mathsf{Cost}^P_o = \begin{cases} \mathit{cost}(o)\mathsf{Y}_P & \text{if } o \text{ affects } P \\ 0 & \text{otherwise} \end{cases} \qquad \text{for all } o \in \mathcal{O} \text{ and } P \in \mathcal{C} \tag{9.17}$$

$$\mathsf{Y}_P \in \{0, 1\} \qquad \text{for all } P \in \mathcal{C} \tag{9.18}$$

$$\sum_{P \in \mathcal{C}} \mathsf{Y}_P = 1 \tag{9.19}$$

The LP with just constraint (9.15) is the LP for optimal general operator cost partitioning over the projection heuristics ($h^{\text{OCP}}_{\mathcal{C}}$). Adding constraint (9.16) then requires the

cost functions to be non-negative ($h_{\mathcal{C}}^{\text{OCP+}}$). When constraint (9.17) is also added, the resulting LP combines the estimate with post-hoc optimization ($h_{\mathcal{C}}^{\text{pho}}$). Constraint (9.18) requires that the factors used by post-hoc optimization are binary, so the MIP for constraints (9.15)–(9.18) computes the canonical heuristic ($h_{\mathcal{C}}^{\text{canon}}$). Finally, with constraint (9.19) only one factor can be 1, so using all constraints amounts to maximization.

9.4. Net Change

The next type of operator-counting constraint we want to introduce here is related to a heuristic originally introduced as *order-relaxed heuristic* (van den Briel et al., 2007). It was later independently derived by Bonet (2013) as the state equation of a Petri net related to the planning task. He called it the *state equation heuristic* h^{SEQ}. Here, we present the general ideas behind h^{SEQ} again by working on the planning task directly, without the translation to Petri nets. This will not only lead us to a deeper understanding but also to a wider class of constraints. In this section we assume planning tasks are in transition normal form (see Section 2.4) to simplify the presentation. Our definitions can be generalized to SAS$^+$ tasks, but we show in Appendix B.1 that the constraint for a general task Π are identical to the constraints introduced here for the transition normalization of Π. Considering only tasks in TNF is therefore no restriction.

We define the *net change* of an atom $\langle V, v\rangle$ from a state s to a state s' as the change of its truth value, where 1 means that the atom becomes true, 0 that it is unchanged, and -1 that it becomes false.

Definition 9.6 (net change)**.** *The* net change *of an atom $\langle V, v\rangle$ from a state s to a state s' is*

$$\begin{aligned} \mathit{netchange}_{\langle V,v\rangle}^{s\to s'} &= \begin{cases} 1 & \text{if } s[V] \neq v \text{ and } s'[V] = v \\ -1 & \text{if } s[V] = v \text{ and } s'[V] \neq v \\ 0 & \text{otherwise} \end{cases} \\ &= [s'[V] = v] - [s[V] = v]. \end{aligned}$$

We say that an operator o applied in state s *produces* an atom $\langle V, v\rangle$ if $s[V] \neq v$ and $s[\![o]\!][V] = v$ and that it *consumes* the atom if $s[V] = v$ and $s[\![o]\!][V] \neq v$. For tasks in TNF it is easy to decide whether an operator produces or consumes an atom and this is independent of the state in which the operator is applied. Each operator o produces all atoms in $\mathit{eff}(o) \setminus \mathit{pre}(o)$ and consumes the atoms in $\mathit{pre}(o) \setminus \mathit{eff}(o)$. All other atoms (including the atoms in $\mathit{pre}(o) \cap \mathit{eff}(o)$) are not modified by o. Obviously, the net change of the atom from state s to the successor state $s[\![o]\!]$ is 1 if o produces the atom, -1 if it consumes it and 0 otherwise. We would like to retain this more operator-centric view:

Definition 9.7 (induced net change)**.** *Let o be an operator and $\pi = \langle o_1, \ldots, o_n \rangle$ an operator sequence such that o and π are applicable in a state s. The* net change that o induces for atom $\langle V, v \rangle$ *is*

$$\mathit{netchange}(o)_{\langle V,v \rangle} = \begin{cases} 1 & \text{if } \langle V, v \rangle \in \mathit{eff}(o) \setminus \mathit{pre}(o) \\ -1 & \text{if } \langle V, v \rangle \in \mathit{pre}(o) \setminus \mathit{eff}(o) \\ 0 & \text{otherwise.} \end{cases}$$

The accumulated net change *induced by sequence π is*

$$\mathit{netchange}(\pi)_{\langle V,v \rangle} = \sum_{i=1}^{n} \mathit{netchange}(o_i)_{\langle V,v \rangle} .$$

It is obvious that we do not need to consider the intermediate states of the application of an operator sequence π but can directly compare the initial and the resulting state:

Proposition 9.2. *When executing an operator sequence π in a state s the net change from s to the resulting state $s[\![\pi]\!]$ is the accumulated net change induced by π:*

$$\mathit{netchange}^{s \to s[\![\pi]\!]}_{\langle V,v \rangle} = \mathit{netchange}(\pi)_{\langle V,v \rangle} .$$

We would like to use this information to derive operator-counting constraints. Since there is a single goal state for tasks in TNF, Proposition 9.2 implies that every plan π induces the accumulated net change $\mathit{netchange}^{s_\mathrm{I} \to s_\star}_{\langle V,v \rangle} = [s_\star[V] = v] - [s_\mathrm{I}[V] = v]$. Using the definitions above, we can rewrite the left-hand side of this equation as follows:

$$\begin{aligned}
\mathit{netchange}^{s_\mathrm{I} \to s_\star}_{\langle V,v \rangle} &= \mathit{netchange}(\pi)_{\langle V,v \rangle} \\
&= \sum_{i=1}^{n} \mathit{netchange}(o_i)_{\langle V,v \rangle} \\
&= \sum_{i=1}^{n} [\langle V, v \rangle \in \mathit{eff}(o_i) \setminus \mathit{pre}(o_i)] - \sum_{i=1}^{n} [\langle V, v \rangle \in \mathit{pre}(o_i) \setminus \mathit{eff}(o_i)] \\
&= \sum_{i=1}^{n} [\langle V, v \rangle \in \mathit{eff}(o_i)] - \sum_{i=1}^{n} [\langle V, v \rangle \in \mathit{pre}(o_i)] \\
&= \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{eff}(o)}} \mathit{occur}_\pi(o) \; - \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{pre}(o)}} \mathit{occur}_\pi(o)
\end{aligned}$$

We define two operator-counting constraints based on the derivation above. It might be surprising that we treat the equation as two inequalities, but this allows us more freedom in their combination which turns out to be useful in this case.

Definition 9.8 (net change constraint)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, cost \rangle$ be a planning task in TNF and s one of its states. For an atom $\langle V, v \rangle$ over $\mathcal{V}$ the* lower bound net change constraint $c^{\mathrm{ncl}}_{s,\langle V,v \rangle}$ *for atom $\langle V, v \rangle$ and state s is the constraint*

$$\sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{eff}(o)}} \mathsf{Count}_o \; - \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{pre}(o)}} \mathsf{Count}_o \geq [s_\star[V] = v] - [s_{\mathrm{I}}[V] = v]$$

and the upper bound net change constraint $c^{\mathrm{ncu}}_{s,\langle V,v \rangle}$ *is the constraint*

$$\sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{eff}(o)}} \mathsf{Count}_o \; - \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{pre}(o)}} \mathsf{Count}_o \leq [s_\star[V] = v] - [s_{\mathrm{I}}[V] = v].$$

The state equation heuristic (Bonet, 2013) is defined via an LP that directly fits our framework. It uses one variable X_o for each operator o and minimizes the same cost function as an operator-counting heuristic.

Definition 9.9 (h^{SEQ}, Bonet, 2013)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, cost \rangle$ be a planning task in TNF with atoms $\mathcal{A}$. The* state equation heuristic h^{SEQ} *maps each state s to the optimal objective value of the following LP or to ∞ if the LP is infeasible.*[3]

$$\textit{Minimize} \sum_{o \in \mathcal{O}} \mathit{cost}(o)\mathsf{X}_o \textit{ subject to}$$

$$\sum_{o \in \mathcal{O}} a_{o,\langle V,v \rangle}\mathsf{X}_o \geq m_{s_\star,\langle V,v \rangle} - m_{s,\langle V,v \rangle} \qquad \textit{for all } \langle V, v \rangle \in \mathcal{A}$$

where $m_{s,\langle V,v \rangle} = [s[V] = v]$ and $a_{o,\langle V,v \rangle} = [\langle V, v \rangle \in \mathit{eff}(o)] - [\langle V, v \rangle \in \mathit{pre}(o)]$ are derived from the state markings and the incidence matrix of the Petri net associated with Π.

A close look at the constraints reveals they are lower bound net change constraints. The variable X_o for operator o occurs in the constraint for atom $\langle V, v \rangle$ with the coefficient $-\mathit{netchange}(o)_{\langle V,v \rangle}$.

Proposition 9.3. *For state s, let $\mathcal{C}(s)$ denote the set of lower bound net change constraints for s and all atoms. Then the state equation heuristic h^{SEQ} equals the LP heuristic $h^{\mathrm{LP}}_{\mathcal{C}}$.*

Bonet also suggests a safety-based improvement and a landmark-based improvement of h^{SEQ}. A variable V is called *safe* if all operators that affect V also have a precondition on V. For tasks in TNF, all variables are safe. For atoms of safe variables the constraints in the LP of h^{SEQ} can be tightened to equations.

[3] Bonet's definition includes rounding up the optimal objective value. Since operator costs are natural numbers, this can be done with any admissible heuristic function (just like maximizing with 0). Here, we view rounding as an optimization, not as part of the definition because this simplifies reasoning about the heuristic. In our implementation, we use $h' = \max(\lceil h \rceil, 0)$ for all heuristics h.

Proposition 9.4. *The safety-based improvement of h^{SEQ} corresponds to extending each set $\mathcal{C}(s)$ with the upper bound net change constraints for s and all atoms of safe variables. The landmark-based improvement of h^{SEQ} corresponds to extending $\mathcal{C}(s)$ with the landmark constraints for the given landmarks.*

9.5. Prevail Conditions

Net change constraints ignore atoms that are required but not modified by an operator. The coefficients of these so-called *prevail conditions* cancel out in the constraints. Van den Briel et al. (2007) suggest handling prevail conditions with constraints that fit the operator-counting framework. They require that a used prevail condition must be initially true or be made true by some operator application:

Definition 9.10 (positive prevail constraint)**.** *Let Π be a planning task and s one of its states. Let o be an operator of Π with a prevail condition on atom $\langle V, v\rangle$. The* positive prevail constraint *for o, $\langle V, v\rangle$, and s is*

$$\sum_{\substack{o' \in \mathcal{O} \setminus \{o\} \\ \langle V, v\rangle \in \mathit{eff}(o')}} \mathsf{Count}_{o'} \geq 1/M \, \mathsf{Count}_o - [s[V] = v].$$

where M is a constant large enough to guarantee $0 < 1/M \, \mathsf{Count}_o \leq 1$ if $\mathsf{Count}_o > 0$.

Note that the constraint is trivially satisfied if $s[V] = v$, so one could in practice simply omit it from the linear program if the prevail condition is already satisfied in s. The constraint is also trivially satisfied if o is not used. In other cases the constraint implies that at least one operator o' that adds the prevail condition has a positive count. Since using o is only possible if the prevail condition is added by some operator, positive prevail constraints are operator-counting constraints.

In a MIP positive prevail constraints imply that an operator that adds the prevail condition has a count of at least 1. However, constraints that are disabled by a large constant in some situations (so-called *big-M* constraints) are notoriously weak in the LP-relaxation (Camm, Raturi, and Tsubakitani, 1990).

We can additionally define constraints that require that if a prevail condition is used in a plan but the atom must be false in the goal then it must be consumed at some point:

Definition 9.11 (negative prevail constraint)**.** *Let Π be a planning task with goal $s_\star$. Let o be an operator of Π with a prevail condition on atom $\langle V, v\rangle$. The* negative prevail constraint *for o and $\langle V, v\rangle$ is*

$$\sum_{\substack{o' \in \mathcal{O} \\ \langle V, v'\rangle \in \mathit{eff}(o') \\ v' \neq v}} \mathsf{Count}_{o'} \geq 1/M \, \mathsf{Count}_o - [s_\star[V] = v' \neq v]$$

where M is a constant large enough to guarantee $0 < 1/M \, \mathsf{Count}_o \leq 1$ if $\mathsf{Count}_o > 0$.

9.6. Network Flow

The final type of operator-counting constraints we introduce here is based on network flow in abstract transition systems. The state equation heuristic has the same model as the order-relaxed heuristic suggested by van den Briel et al. (2007). The latter is derived from network flows in domain transition graphs (DTGs) which are closely related to atomic projections. The network flow constraints we introduce here can thus be seen as a generalization of net change constraints and the state equation heuristic. We will explore this connection in more detail in Section 10.4, but first we introduce network flow and the constraints formally. For this analysis, we again assume that the tasks are in TNF. In Appendix B.2 we discuss flow constraints for general SAS^+ tasks.

Consider a transition system $\text{TS} = \langle \mathcal{S}, \mathcal{T}, s_\text{I}, \{s_\star\} \rangle$ with a single goal state and transition labels $\mathcal{O}$. For a state $s \in \mathcal{S}$ we write $\mathit{in}_\text{TS}(s)$ and $\mathit{out}_\text{TS}(s)$ for the set of transitions that end and start in s, and write $\mathit{trans}_\text{TS}(o)$ to denote the set of transitions labeled with o. A *flow* in TS maps each transition t to a non-negative real number called the *flow along* t, such that the total flow along incoming transitions matches the total flow along outgoing transitions in each node, except at the initial and goal nodes. The cost of a flow is the summed cost of each transition multiplied by its flow. As we want to use the cost of a flow in the context of general cost partitioning, we allow the cost function to take all real values.

The following LP is the standard formulation of a minimum-cost flow problem that "moves" one unit of flow from the source node s_I to the sink node $s_\star$ in TS:

$$\text{Minimize} \sum_{o \in \mathcal{O}} \sum_{t \in \mathit{trans}_\text{TS}(o)} \mathit{cost}(o) \mathsf{Count}_t \text{ subject to}$$

$$\sum_{t \in \mathit{in}_\text{TS}(s)} \mathsf{Count}_t - \sum_{t \in \mathit{out}_\text{TS}(s)} \mathsf{Count}_t = [s = s_\star] - [s = s_\text{I}] \qquad \text{for all } s \in \mathcal{S} \tag{9.20}$$

$$\mathsf{Count}_t \geq 0 \qquad \text{for all } t \in \mathcal{T} \tag{9.21}$$

where $[s = s_\star] - [s = s_\text{I}]$ represents the amount of flow "consumed" at state s; i.e. one unit produced at s_I, one unit consumed at $s_\star$, and zero units at other states. In this LP, the flow along a transition t is denoted by Count_t.

We call this LP the *flow model* for TS and s_I, and call the constraint (9.20) the *flow balance equation* for s. Flow balance equations are very similar to net change constraints, but there are two important differences: firstly, flow balance equations consider abstract states and transitions, whereas net change constraints consider atoms and operators. Secondly, and more fundamentally, flow balance equations use one LP variable for each abstract transition, while net change constraints only use one LP variable for each operator. We analyze the differences between the two constraints in Section 10.4.

It can be shown that the flow model is feasible iff there is a path from s_I to $s_\star$ in TS (Korte and Vygen, 2001). If costs are non-negative, feasible solutions always have bounded cost, but if the LP is feasible and there is a cycle of transitions in TS with

negative total cost (not necessarily connected to the initial or goal state), the LP is unbounded and its value is $-\infty$ because an unbounded amount of flow may circulate through the negative cost cycle.

Definition 9.12 (flow heuristic)**.** *Let α be an abstraction of a TNF planning task Π. The* flow heuristic *f^α maps each state s of Π to the optimal objective value of the flow model for the transition system TS^α of α and s, or to ∞ if the model is infeasible.*

The flow heuristic f^α is closely related to the abstraction heuristic h^α:

Proposition 9.5. *Let α be an abstraction of a TNF planning task Π and $cost_\alpha : \mathcal{O} \to \mathbb{R}$ a general cost function. Then, $f^\alpha(s, cost_\alpha) \leq h^\alpha(s, cost_\alpha)$ for all states s with equality if there are no dead states in TS^α.*

Proof: As mentioned before, $h^\alpha(s, cost_\alpha) = \infty$ iff there is no path in TS^α from $\alpha(s)$ to $\alpha(s_\star)$ iff $f^\alpha(s, cost_\alpha) = \infty$.

Assume now that $h^\alpha(s, cost_\alpha) < \infty$. For a path in TS^α from $\alpha(s)$ to $\alpha(s_\star)$ with cost c, there is a flow with the same cost. Therefore, $f^\alpha(s, cost_\alpha) \leq h^\alpha(s, cost_\alpha)$ (this includes the case where a negative cost cycle occurs on an alive state and $h^\alpha = f^\alpha = -\infty$).

If $h^\alpha(s, cost_\alpha)$ is finite and there are no dead states in TS^α, no negative cost cycles exist in TS^α and thus $f^\alpha(s, cost_\alpha)$ is also finite. The flow along all transitions may be assumed to be integral (Korte and Vygen, 2001). This flow moves one unit from $\alpha(s)$ to $\alpha(s_\star)$ along a single minimum cost path. Therefore, $h^\alpha(s, cost_\alpha) = f^\alpha(s, cost_\alpha)$. □

The flow heuristic can be seen as an operator-counting heuristic. Indeed, it is enough to modify the flow model by adding the operator-counting variables Count_o for each operator o, replacing the objective function by $\sum_{o \in \mathcal{O}} cost(o)\mathsf{Count}_o$, and adding constraints $\sum_{t \in trans_{\mathrm{TS}}(o)} \mathsf{Count}_t = \mathsf{Count}_o$ for each operator o. We call each such constraint the *strong linking constraint* for o. Together, flow balance equations and strong linking constraints form an operator-counting constraint.

Definition 9.13 (flow constraint)**.** *Let α be an abstraction of a TNF planning task Π and let $\mathrm{TS}^\alpha = \langle \mathcal{S}^\alpha, \mathcal{T}^\alpha, s^\alpha_{\mathrm{I}}, \{s^\alpha_\star\} \rangle$ be its abstract transition system. For a state s of Π, the* flow constraint *$c^{\mathrm{flow}}_{s,\alpha}$ for α and s uses non-negative* transition-counting variables *Count_t for every abstract transition $t \in \mathcal{T}^\alpha$ as auxiliary variables. It consists of the flow balance equation for each abstract state $s' \in \mathcal{S}^\alpha$:*

$$\sum_{t \in in_{\mathrm{TS}^\alpha}(s')} \mathsf{Count}_t - \sum_{t \in out_{\mathrm{TS}^\alpha}(s')} \mathsf{Count}_t = [s' = s^\alpha_\star] - [s' = \alpha(s)]$$

and a strong linking constraint for each operator $o \in \mathcal{O}$:

$$\sum_{t \in trans_{\mathrm{TS}^\alpha}(o)} \mathsf{Count}_t = \mathsf{Count}_o.$$

Every s-plan of Π induces a path from $\alpha(s)$ to $s^\alpha_\star$ in TS^α, which in turn induces a valid flow in TS^α with the same number of operator occurrences. Thus, $c^{\mathrm{flow}}_{s,\alpha}$ is indeed an operator-counting constraint. The LP heuristic for these constraints corresponds to the flow heuristic f^α:

Proposition 9.6. *Let α be an abstraction of a TNF planning task Π and s one of its states. Then,*

$$h^{\mathrm{LP}}_{c^{\mathrm{flow}}_{s,\alpha}}(s) = f^\alpha(s).$$

Proof: The model of the operator-counting LP heuristic $h^{\mathrm{LP}}_{c^{\mathrm{flow}}_{s,\alpha}}$ is just a slight reformulation of the flow model for TS^α and s as mentioned above. Both models have the same optimal objective value and their solutions are in 1:1 correspondence. □

10. Theoretical Analysis

In the previous chapter we introduced several types of operator-counting constraints, demonstrating that heuristics of different origins fit the framework. On the one hand this offers a way to *combine* heuristics in a smarter way than using their maximum. On the other hand, it allows us to *reason* about heuristics in a common language. In this chapter, we provide examples of such reasoning and analyze the heuristic combination in more detail.

We first look deeper into the combination of multiple operator-counting constraints and prove an interesting connection to cost partitioning. This connection offers new perspectives on known heuristics, which we discuss next. We then show that certain inequalities of h^{SEQ} are redundant and how flow heuristics are connected to this. Finally, we explore the limits of operator counting and prove that critical path heuristics can not be expressed with operator-counting constraints in a useful way. This answers the question why we did not introduce an operator-counting constraint based on them in Chapter 9.

10.1. Connection to General Cost Partitioning

In Part I we saw that cost partitioning can be used to admissibly combine heuristic functions. Operator-counting heuristics also allow such a heuristic combination, as we saw in Chapter 8. Both techniques can have synergy effects, where the combination results in a higher heuristic value than the maximum of its components, but how are they related to each other? We now answer that question for operator-counting LP heuristics and general operator cost partitioning.

For a given constraint set, the value of an operator-counting heuristic depends only on the operator costs. We make this explicit by writing $h^{\text{LP}}_{\mathcal{C}}(\mathit{cost})$. Given a constraint set of operator-counting constraints, its LP heuristic estimate equals the optimal general operator cost partitioning over the LP heuristics for each *individual* constraint:

Theorem 10.1. *Let $\mathcal{C}$ be a constraint set for a state s. Then*

$$h^{\text{LP}}_{\mathcal{C}}(\mathit{cost}) = h^{\text{OCP}}_{\{h^{\text{LP}}_{\{C\}} \mid C \in \mathcal{C}\}}(s).$$

Proof sketch: Operator-counting constraints may use auxiliary variables. While these do not make the proof much harder, they require more complex notation and generally

distract from the core idea of the proof. We thus ignore them here and defer the full proof to Appendix A.

The linear program solved by $h^{\mathrm{LP}}_{\mathcal{C}}(\mathit{cost})$ can be written in matrix notation as follows.

$$\begin{aligned} &\text{Minimize } \sum_{o \in \mathcal{O}} \mathit{cost}(o)\mathsf{Count}_o \text{ subject to} \\ \mathit{count\text{-}coeffs}^C \mathbf{Count} &\geq \mathit{bounds}^C && \text{for } C \in \mathcal{C} \\ \mathbf{Count} &\geq \mathbf{0} \end{aligned}$$

Here, $\mathit{count\text{-}coeffs}^C$ is the matrix of coefficients and bounds^C is the column vector of bounds used by operator-counting constraint C.

In this linear program, we introduce *local copies* of the operator-counting variables, as new non-negative variables LCount^C_o and new equations $\mathsf{LCount}^C_o = \mathsf{Count}_o$ for every $C \in \mathcal{C}$ and $o \in \mathcal{O}$. Replacing every occurrence of Count_o in the remaining inequalities by LCount^C_o does not influence the optimal objective value.

$$\begin{aligned} &\text{Minimize } \sum_{o \in \mathcal{O}} \mathit{cost}(o)\mathsf{Count}_o \text{ subject to} \\ \mathit{count\text{-}coeffs}^C \mathbf{LCount} &\geq \mathit{bounds}^C && \text{for } C \in \mathcal{C} && (10.1) \\ \mathbf{Count} - \mathbf{LCount}^C &= \mathbf{0} && \text{for } C \in \mathcal{C} && (10.2) \\ \mathbf{LCount}^C &\geq \mathbf{0} && \text{for } C \in \mathcal{C} && (10.3) \\ \mathbf{Count} &\geq \mathbf{0} && && (10.4) \end{aligned}$$

The dual of this LP has dual variables for each constraint 10.1 which we call $\mathbf{Dual}^C$ and dual variables for each constraint 10.2 which we call $\mathbf{Cost}^C$:

$$\begin{aligned} &\text{Maximize } \sum_{C \in \mathcal{C}} {\mathit{bounds}^C}^\top \mathbf{Dual}^C \text{ subject to} \\ {\mathit{count\text{-}coeffs}^C}^\top \mathbf{Dual}^C &\leq \mathbf{Cost}^C && \text{for } C \in \mathcal{C} \\ \sum_{C \in \mathcal{C}} \mathsf{Cost}^C_o &\leq \mathit{cost}(o) && \text{for } o \in \mathcal{O} \\ \mathbf{Dual}^C &\geq \mathbf{0} && \text{for } C \in \mathcal{C} \end{aligned}$$

This is an optimal general operator cost partitioning over the heuristics h^C where h^C under a cost function cost^C is defined as the optimal objective value of the following LP:

$$\begin{aligned} &\text{Maximize } {\mathit{bounds}^C}^\top \mathbf{Dual}^C \text{ subject to} \\ {\mathit{count\text{-}coeffs}^C}^\top \mathbf{Dual}^C &\leq \mathit{cost}^C \\ \mathbf{Dual}^C &\geq \mathbf{0} \end{aligned}$$

The dual of this LP is exactly the LP solved by $h^{\text{LP}}_{\{C\}}(\mathit{cost}^C)$. Therefore, $h^C = h^{\text{LP}}_{\{C\}}$ and $h^{\text{LP}}_{\mathcal{C}}(\mathit{cost}) = h^{\text{OCP}}_{\{h^C|C\in\mathcal{C}\}}(s) = h^{\text{OCP}}_{\{h^{\text{LP}}_{\{C\}}|C\in\mathcal{C}\}}(s)$. □

The proof is constructive in the sense that it shows a way to compute an optimal cost partitioning for the given state from a dual solution of the operator-counting LP.

Theorem 10.2. *Let $\mathcal{C}$ be a constraint set for a state s. Let d be a dual solution of* $\text{LP}_{\mathcal{C}}(\mathit{cost})$. *Then the general cost partitioning*

$$\mathit{cost}^C(o) = \mathit{count\text{-}coeffs}^{C^\top} d(\mathbf{Dual}^C) \quad \text{for } C \in \mathcal{C}$$

is optimal for the heuristics $h^{\text{LP}}_{\{C\}}$ for $C \in \mathcal{C}$ in state s.

Proof: Every optimal solution of the LP in the proof for Theorem 10.1 contains an optimal cost partition in the variables Cost^C_o. If we take an optimal solution and replace the value of Cost^C_o with $\mathit{count\text{-}coeffs}^{C^\top} d(\mathbf{Dual}^C)$, no value of Cost^C_o can increase. All inequalities are still satisfied and the optimal objective value does not change. □

Theorems 10.1 and 10.2 establish a connection between cost partitioning and operator counting. This connection can be used in two ways. First, it gives an interpretation of operator-counting heuristics that use constraints from different sources. For example, an operator-counting LP heuristic using lower bound net change constraints for all atoms and delete-relaxation constraints can be seen as combining h^{SEQ} and the LP relaxation of h^+ with optimal operator cost partitioning. A second way to use the theorem is to analyze existing heuristics that can be expressed in the operator-counting framework and gain new insight by interpreting them as a cost partitioning of several component heuristics. For such an analysis, we consider a constraint set $\mathcal{C}$ for some state and find an interpretation of the heuristics $h^{\text{LP}}_{\{C\}}(\mathit{cost}')$ for all $C \in \mathcal{C}$, i.e. the operator-counting LP heuristics for individual constraints under a general cost function cost'. Theorem 10.1 then allows us to interpret $h^{\text{LP}}_{\mathcal{C}}$ as the optimal cost partitioning of the component heuristics.

An analysis with Theorem 10.1 gives a new perspective on heuristics that are equivalent to an operator-counting LP heuristic over multiple operator-counting constraints, such as the state equation heuristic. It does not help if the heuristic is based on a single operator-counting constraint. The delete-relaxation heuristic h^+, for example, can be expressed as the operator-counting MIP heuristic over the delete-relaxation constraint. The LP relaxation could be interpreted with Theorem 10.1 but the heuristic only uses a single operator-counting constraint. Even though it consists of several inequalities, the constraint cannot be easily split into more than one operator-counting constraint because such constraints may not share auxiliary variables. For the same reason, flow constraints cannot be split into smaller parts.

10.2. Analyzing Landmark Heuristics

Let us first consider landmark constraints. If L is a landmark for state s, the operator-counting LP for the landmark constraint $c^{\text{lm}}_{s,L}$ under a cost function *cost'* is:

$$\text{Minimize} \sum_{o \in \mathcal{O}} \mathit{cost}'(o)\mathsf{Count}_o \text{ subject to}$$
$$\sum_{o \in L} \mathsf{Count}_o \geq 1$$

For every choice of L and *cost'* this LP has the optimal objective value $\min_{o \in L} \mathit{cost}'(o)$. Since this is the cost of L under *cost'*, we can use Theorem 10.1 to show the following proposition.

Proposition 10.1. *Let $\mathcal{L}$ be a set of landmarks for a state s and let $\mathcal{C}$ be the constraint set $\{c^{\text{lm}}_{s,L} \mid L \in \mathcal{L}\}$. Then $h^{\text{LP}}_{\mathcal{C}}$ is the optimal general operator cost partitioning over the landmarks in $\mathcal{L}$.*

We already mentioned a similar result by Bonet and Helmert (2010) for non-negative cost partitioning. Together with the proposition above, this implies the following corollary.

Corollary 10.1. *Allowing negative costs in the cost partitioning of landmarks cannot increase the heuristic value.*

An alternative way to see this is to observe that all coefficients in the operator-counting LP are non-negative. According to Theorem 10.2 an optimal cost partitioning can be computed as the matrix product of a dual solution and the coefficients. The dual variables can only take non-negative values, so if all coefficients are also non-negative, there has to be a non-negative cost partitioning that is also optimal among general cost partitionings. For the same reason, general costs cannot increase the heuristic value if only post-hoc optimization constraints are used.

10.3. Analyzing the State Equation Heuristic as a Net Change Heuristic

We now turn to the state equation heuristic h^{SEQ}, which can be approached from two directions. As shown in Proposition 9.3, the state equation heuristic is the operator-counting LP heuristic for the set of lower bound net change constraints for all atoms. We use this connection to analyze the safety-based improvement next. Afterwards, we explore an alternative view on h^{SEQ} via flow heuristics.

In the paper that introduced the state equation heuristic, Bonet (2013) also suggested the previously-mentioned extension exploiting safe state variables and reported a (modest) improvement in performance when using it. From Proposition 9.4 we know that

the safety-based extension corresponds to adding upper bound net change constraints to the LP. We now show that these constraints cannot improve the heuristic beyond the basic h^{SEQ} estimate.

The proof is limited to tasks in TNF but the results also hold for unrestricted SAS$^+$ tasks, where the definitions of net change constraints have to be suitably extended. As this generalization complicates the presentation without adding much insight, we discuss it in Appendix B.1. To simplify notation, we introduce functions for both sides of a net change constraint. For a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\text{I}}, s_\star, \mathit{cost} \rangle$ in TNF, we define:

$$\mathit{lhs}_{\langle V,v \rangle}(\mathsf{Count}) = \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{eff}(o)}} \mathsf{Count}_o \; - \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{pre}(o)}} \mathsf{Count}_o$$
$$\mathit{rhs}_{\langle V,v \rangle} = [s_\star[V] = v] - [s_{\text{I}}[V] = v].$$

The the lower bound net change constraints for $\langle V, v \rangle$ is $\mathit{lhs}_{\langle V,v \rangle}(\mathsf{Count}) \geq \mathit{rhs}_{\langle V,v \rangle}$ and the upper bound net change constraint is $\mathit{lhs}_{\langle V,v \rangle}(\mathsf{Count}) \leq \mathit{rhs}_{\langle V,v \rangle}$.

We first consider the left-hand side of a net change constraint and show that the total net change induced by a plan over all values of a variable is 0. Intuitively, this is the case because, whenever one value is produced, another value of the same variable is consumed.

Operators in TNF mention the same variables in their preconditions and effects, and since $\mathit{pre}(o)$ and $\mathit{eff}(o)$ are partial states, they cannot contain two atoms with variable V. We thus have

$$\sum_{v \in \mathit{dom}(V)} \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{eff}(o)}} \mathsf{Count}_o = \sum_{\substack{o \in \mathcal{O} \\ V \in \mathit{vars}(o)}} \mathsf{Count}_o = \sum_{v \in \mathit{dom}(V)} \sum_{\substack{o \in \mathcal{O} \\ \langle V,v \rangle \in \mathit{pre}(o)}} \mathsf{Count}_o.$$

This implies for all variables V and all values of Count

$$\sum_{v \in \mathit{dom}(V)} \mathit{lhs}_{\langle V,v \rangle}(\mathsf{Count}) = 0. \tag{10.5}$$

Now consider the right-hand side of the constraint. In TNF there is exactly one initial and one goal state, so

$$\sum_{v \in \mathit{dom}(V)} [s_\star[V] = v] = 1 = \sum_{v \in \mathit{dom}(V)} [s_{\text{I}}[V] = v].$$

This implies

$$\sum_{v \in \mathit{dom}(V)} \mathit{rhs}_{\langle V,v \rangle} = 0. \tag{10.6}$$

Proposition 10.2. *Let Π be a TNF planning task and $\langle V, v\rangle$ one of its atoms. Then*

$$\sum_{\substack{v' \in dom(V)\\ v' \neq v}} lhs_{\langle V,v'\rangle}(\mathsf{Count}) \geq \sum_{\substack{v' \in dom(V)\\ v' \neq v}} rhs_{\langle V,v'\rangle} \quad \textit{iff} \quad lhs_{\langle V,v\rangle}(\mathsf{Count}) \leq rhs_{\langle V,v\rangle}.$$

Proof: The statement follows from (10.5) and (10.6) with basic arithmetic. □

Theorem 10.3. *Let Π be a TNF planning task and V one of its variables. Every feasible solution of the set of all lower bound net change constraints for atoms of V is also feasible for the set of all upper bound net change constraints for atoms of V.*

Proof: In general, any solution of a set of constraints $\mathbf{c_i x} \geq \mathbf{b_i}$ for $i \in I$ also satisfies the constraint $(\sum_{i\in I} \mathbf{c_i})\mathbf{x} \geq \sum_{i\in I} \mathbf{b_i}$. Summing up all lower bound net change constraints for variable V and values $dom(V) \setminus \{v\}$ in this way is equivalent to the upper bound net change constraint for $\langle V, v\rangle$ according to Proposition 10.2. □

This result challenges the safety-based improvement of the state equation heuristic. The experimental benefit reported for the safety-based improvement must hence have a different cause, such as faster heuristic computation or noise.

Corollary 10.2. *The safety-based improvement of the state equation heuristic cannot improve the heuristic estimates.*

10.4. Analyzing the State Equation Heuristic as a Flow Heuristic

The state equation heuristic h^{SEQ} can be defined over flows in domain transition graphs (DTGs) (van den Briel et al., 2007; Bonet and van den Briel, 2014). A DTG for a variable V in a TNF task is a directed graph with one node for every value of V and an edge $v \xrightarrow{o} v'$ for every operator o with $pre(o)[V] = v$ and $\mathit{eff}(o)[V] = v'$. The *DTG constraint* for atom $\langle V, v\rangle$ is

$$\sum_{v' \xrightarrow{o} v \in in_{\text{DTG}}(v)} \mathsf{Count}_o \; - \sum_{v \xrightarrow{o} v' \in out_{\text{DTG}}(v)} \mathsf{Count}_o = [s_\star[V] = v] - [s_{\text{I}}[V] = v].$$

It is easy to see that the DTG constraint for an atom is equivalent to the upper and lower bound net change constraints for the atom. According to Proposition 9.3, the state equation heuristic can then be seen as the operator-counting LP heuristic with DTG constraints for all atoms. The definition by Bonet and van den Briel (2014) uses inequalities, but with safety-based improvement all of them turn into equations in TNF tasks.

An operator-counting constraint can consist of any number of linear inequalities. For our purposes, it is useful to group the DTG constraint for all atoms that belong to the same state variable V into one constraint C_V. We prefer the representation based on DTGs and with the safety-based improvement here because it is closer to flow constraints. Extensions to h^{SEQ} consider additional constraints, which we will describe as they become relevant.

As a reminder, $c^{\text{flow}}_{s,\alpha}$, the flow constraint for an abstraction α and state s consists of the flow balance equation for each abstract state $s' \in \mathcal{S}^\alpha$ of the transition system TS^α:

$$\sum_{t \in in_{\text{TS}^\alpha}(s')} \mathsf{Count}_t \quad - \sum_{t \in out_{\text{TS}^\alpha}(s')} \mathsf{Count}_t = [s' = s^\alpha_\star] - [s' = \alpha(s)]$$

and a strong linking constraint for each operator $o \in \mathcal{O}$:

$$\sum_{t \in trans_{\text{TS}^\alpha}(o)} \mathsf{Count}_t = \mathsf{Count}_o.$$

We will show that all extensions of the state equation heuristic can be understood as simplified versions of this model. This is interesting because it shows a connection to cost-partitioned abstractions.

Proposition 10.3. *Let A be a set of abstractions of a planning task and for every abstraction $\alpha \in A$ let $\mathcal{C}^\alpha$ be the function that maps every state s to $\{c^{\text{flow}}_{s,\alpha}\}$. Then,*

$$h^{\text{LP}}_{\{\mathcal{C}^\alpha | \alpha \in A\}} = h^{\text{OCP}}_{\{f^\alpha | \alpha \in A\}} \leq h^{\text{OCP}}_{\{h^\alpha | \alpha \in A\}}$$

with equality if, for all $\alpha \in A$, TS^α contains no dead state.

Proof: With Proposition 9.6 we can interpret $h^{\text{LP}}_{\{\mathcal{C}^\alpha\}}$ as f^α and Theorem 10.1 then shows that the LP heuristic computes an optimal cost partitioning over such flow heuristics. With Proposition 9.5 we get the connection to cost-partitioned abstraction heuristics. □

To see that $h^{\text{OCP}}_{\{f^\alpha | \alpha \in A\}}$ may indeed lead to a lower value in the presence of dead states, consider the task in Figure 10.1 (repeated from Section 5.1). The task has a single operator o and two projections α_1 and α_2 to V_1 and V_2. In TS^{α_1} o induces a transition from the initial state to the goal. In TS^{α_2} the initial and goal state are the same, and o induces a self-loop on an unreachable state. For an arbitrarily large M, let $cost_1(o) = M$ and $cost_2(o) = -M$. The shortest paths under these cost functions have cost M and 0, so $h^{\text{OCP}}_{\{h^\alpha | \alpha \in A\}} = \infty$. The minimal flows have cost M and $-\infty$, which is not optimal. The best cost partitioning for the flows uses costs of 1 in α_1 and 0 in α_2 for a total value of $h^{\text{OCP}}_{\{f^\alpha | \alpha \in A\}} = 1$.

10.4.1. Improving the Flow Constraint

We now show how the flow constraint $c^{\text{flow}}_{s,\alpha}$ can be simplified and strengthened by introducing a set of transformation rules. Using them, we show how the state equation heuristic relates to an optimal cost partitioning over certain abstractions.

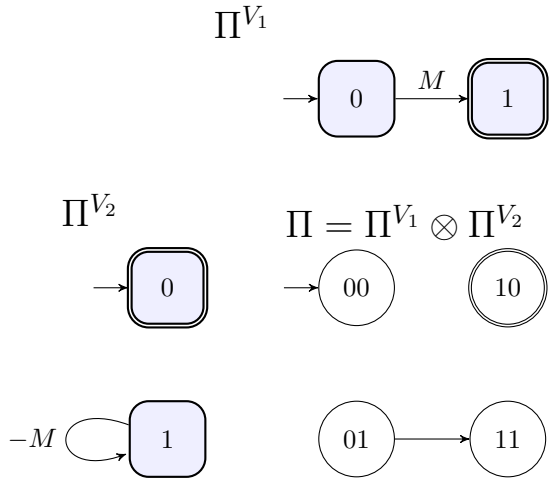


Figure 10.1: Example task Π with binary variables V_1 and V_2. Above and to the left of the original task we show the projections α_1 and α_2 to V_1 and V_2. An optimal cost partitioning over flow heuristics has a lower value than an optimal cost partitioning over abstraction heuristics for the same abstractions: $h^{\text{OCP}}_{\{f^\alpha|\alpha\in A\}} = 1 < \infty = h^{\text{OCP}}_{\{h^\alpha|\alpha\in A\}}$.

Dead States

Dead states cannot be part of a shortest path in any abstraction. Removing such states is an obvious step.

Rule 1. *Remove the flow balance equations for all dead states and all transition-counting variables for transitions adjacent to a dead state. This may strengthen the flow constraint.*

In the context of operator counting, a flow constraint strengthened in this way behaves like a shortest path model. To show this, we have to consider it under general cost functions.

Proposition 10.4. *Let α be an abstraction of a TNF planning task Π. For a state s let $\mathcal{C}(s)$ be the constraint $c^{\text{flow}}_{s,\alpha}$ strengthened with Rule 1. Then $h^{\text{LP}}_{\mathcal{C}}(s, cost') = h^\alpha(s, cost')$ for cost functions $cost' : \mathcal{O} \to \mathbb{R}$.*

Proof: Let TS$'$ be the transition system TS$^\alpha$ without transitions adjacent to dead states. Let f' and h' be the cost of a minimal flow and a shortest path from $\alpha(s)$ to $\alpha(s_\star)$ in TS$'$ under cost function $cost'$. Proposition 9.6 shows $h^{\text{LP}}_{\mathcal{C}}(s, cost') = f'$. Because TS$^\alpha$ and TS$'$ have the same goal paths, $h' = h^\alpha(s, cost')$. Proposition 9.5 establishes the missing link $f' = h'$. □

Operators Inducing a Single Transition

A common situation when considering small projections is that an operator only induces a single transition. In this case the linking constraint $\sum_{t \in trans_{\mathrm{TS}}(o)} \mathsf{Count}_t = \mathsf{Count}_o$ trivializes to $\mathsf{Count}_{t_o} = \mathsf{Count}_o$ for some transition t_o. We can reduce the size of the model by using Count_o directly:

Rule 2. *If an operator o only induces a single transition t_o in an abstraction, replace Count_{t_o} with Count_o in all constraints. Then remove the linking constraint for o and the variable Count_{t_o}. This does not affect the solutions of operator-counting variables in the flow constraint.*

Self Loops

Self-looping transitions cancel out in flow balance equations because they are incoming and outgoing transitions of the same state. Thus, their transition-counting variables only occur in linking constraints. We can use different simplifications depending on how many self-loops and state-changing transitions an operator o induces.

If o induces no state-changing transitions, its transition-counting variables only occur in the linking constraint. But for every value of Count_o the constraint can always be satisfied by setting Count_t to Count_o for some transition t of o and to 0 for all others.

Rule 3. *If an operator o induces at least one self-loop and no state-changing transition, remove the linking constraint for o and all transition-counting variables for transitions labeled with o. This does not affect the solutions of operator-counting variables in the flow constraint.*

DTG constraints for a variable V are based on the DTG of V, which only contains transitions for operators that affect V. All other operators induce a self-loop on all abstract states in the projection to V. Except for these self-loops the transition system of the DTG matches that of a projection to V, so the following proposition is easy to verify.

Proposition 10.5. *The set C_V of DTG constraints for all atoms of a variable V is identical to $c^{\mathrm{flow}}_{s,\alpha}$ for the projection α on V simplified using Rules 2 and 3.*

We can now interpret the operator-counting heuristic $h^{\mathrm{LP}}_{\{C_v\}}$ that uses DTG constraints for all atoms of only one variable.

Corollary 10.3. *Let Π be a planning task and let V be one of its state variables. Let h^V denote the* atomic abstraction heuristic *for the projection α on $\{V\}$. If TS^α contains no dead states (i.e. every value is reachable in the projection to V and a goal value can be reached from every value in the projection), then*

$$h^{\mathrm{LP}}_{\{C_V\}}(\mathit{cost}') = h^V(s, \mathit{cost}')$$

The restriction to reachable and relevant values is no issue in practice, as unreachable and irrelevant values can always be removed from a variable domain in a preprocessing step without affecting plan existence.

We can now specify the connection between the state equation heuristic and abstraction heuristics more precisely:

Theorem 10.4. *The state equation heuristic is the optimal general operator cost partitioning of all atomic projections as long as no atomic projection contains a dead state,*

$$h^{\text{SEQ}}(s) = h^{\text{OCP}}_{\{h^V \mid V \in \mathcal{V}\}}(s).$$

Proof: We apply Theorem 10.1 with the set of constraints $\mathcal{C} = \{C_V \mid V \in \mathcal{V}\}$. Corollary 10.3 establishes the connection to projections and Proposition 9.3 shows that $h^{\text{SEQ}}(s) = h^{\text{LP}}_{\mathcal{C}}(\mathit{cost})$. □

This answers a question raised by Bonet (2013): how does h^{SEQ} relate to the four families of heuristics identified by Helmert and Domshlak (2009)? We now see that it is a general operator cost partitioning over abstraction heuristics if dead values are removed from variable domains. Without removing such values, it is a cost partitioning over flow heuristics which are slightly weaker. We also note that with Theorem 10.2 we can extract an optimal cost partitioning from the dual solution of the LP for h^{SEQ}.

With Rules 2 and 3 we can remove the flow constraint if an operator induces only state-changing transitions or only self-loops. If o induces both self-loops *and* state-changing transitions, then the linking constraint cannot be removed but it can be simplified. We can think of the linking constraint as two inequalities $\sum_{t \in \mathit{trans}_{\text{TS}}(o)} \mathsf{Count}_t \leq \mathsf{Count}_o$ and $\sum_{t \in \mathit{trans}_{\text{TS}}(o)} \mathsf{Count}_t \geq \mathsf{Count}_o$, where only one of them can be unsatisfied at a time. The latter can always be satisfied if a transition t_o only occurs in the linking constraint (i.e. if t_o is a self-loop) by setting Count_{t_o} high enough. In the other inequality, there is no need to mention the counting variables for self-loops. Any solution that assigns a positive flow to them still is a solution if their flow is reduced to 0.

Rule 4. *If an operator o induces at least one self-loop and at least one state-changing transition, replace the strong linking constraint for o with the* weak linking constraint *for o:*

$$\sum_{\substack{t \in \mathit{trans}_{\text{TS}}(o) \\ t \text{ is no self-loop}}} \mathsf{Count}_t \leq \mathsf{Count}_o$$

and remove all transition-counting variables for self-loops labeled with o. This does not affect the solutions of operator-counting variables in the flow constraint.

Bonet and van den Briel (2014) consider merging two variables X and Y into a new variable Z (called the *merge of X and Y*) and introducing constraints for the DTG of Z. The DTG of Z is defined as the *parallel composition* (Dräger, Finkbeiner, and Podelski, 2006) of the DTGs of X and Y where states violating mutexes are removed. We want

to consider mutexes separately, so for now we assume that such states are not removed, i.e. the nodes in the DTG of Z are $dom(X) \times dom(Y)$. Since both X and Y have a goal value and are safe, Z also has a goal value and is safe, so the constraints have the same bounds as in the single-variable case. The only difference is that an operator can induce more than one transition in the DTG of Z. To accurately represent this in the constraints, Bonet and van den Briel introduce an *action copy* for each transition in the DTG and add a constraint to link them to the operator-counting variables. We write the LP variable that counts occurrences of the action copy for transition t as the transition-counting variable Count_t. The constraint that links them to the operator-counting variables then is the weak linking constraint and the constraints introduced for the values of Z are the flow balance equations for the projection on $\{X, Y\}$.

We can see from Rules 3 and 4 that self-loops can be ignored or used to weaken the linking constraint in some cases. However, the constraints generated by the state equation heuristic when merging variables still differ from the flow constraints after using these two rules. Operators inducing only state-changing transitions use a strong linking constraint in the flow constraints and a weak linking constraint in the model of the state equation heuristic. Obviously, a strong linking constraint implies the weak linking constraint. We have equivalence if we minimize $\sum_{o \in \mathcal{O}} cost(o)\mathsf{Count}_o$, all costs are non-negative, and we consider a single abstraction. However, in the context of general cost partitioning, using the strong linking constraint can make a difference for operators that cannot induce self-loops. The constraints for the merge of X and Y in h^{SEQ} can be strengthened by using strong linking constraints for such operators. (Whether an operator only induces state-changing transitions in a projection can be checked syntactically: an operator o should use a strong linking constraint iff there is an affected variable V in the projection with $pre(o)[V] \neq \mathit{eff}(o)[V]$.)

For example, consider a task with two operators $\{o_1, o_2\}$ both with cost 1. Say a merge of two variables implies that o_1 has to be used exactly two times: $\mathsf{Count}_{o_1} = 2$. Now consider a second merge where the DTG contains two paths from the initial state to the goal: $\pi_1 = \langle o_1 \rangle$ and $\pi_2 = \langle o_1, o_2, o_1 \rangle$. With a weak linking constraint for o_1 counting only the transitions of π_1 satisfies all constraints for a total heuristic value of 2. With a strong linking constraint, we are forced to consider only plans with two transitions labeled with o_1, e.g. π_2. In this example, this implies that o_2 is used as well, which results in a heuristic value of at least 3.

Proposition 10.6. *Let $\mathcal{C}^{X,Y}$ be the constraints generated by h^{SEQ} when merging variables X and Y, where weak linking constraints are replaced by strong linking constraints for operators that only induce state-changing transitions in the projection to $\{X, Y\}$. Then $\mathcal{C}^{X,Y}$ is $c^{\text{flow}}_{s,\alpha}$ for the projection to $\{X, Y\}$ simplified with Rules 3 and 4.*

Mutex Information

Removing nodes that violate mutexes (Helmert, 2009; Alcázar and Torralba, 2015) from abstractions is a well-known technique called *constrained abstraction* (Haslum,

Bonet, and Geffner, 2005). States violating a mutex condition are similar to unreachable states. While they can lie on a path in the abstraction, no concrete plan visits a state that is abstracted to them. Removing them and their adjacent transitions cannot decrease the heuristic value. The example after Proposition 10.3 can be adapted to show that removing such states can strengthen the flow constraint by making the unreachable state in the example reachable in the abstraction but mutex violating.

Rule 5. *Remove the flow balance equations for all states that violate mutex conditions and all transition-counting variables for adjacent transitions. This may strengthen the flow constraint.*

The result from Proposition 10.6 can be extended to using mutex information. After merging variables X and Y into Z, the DTG of Z with mutexes removed is equal to the constrained projection to $\{X, Y\}$ except for self-loops of operators that do not mention X and Y.

Proposition 10.7. *Let $\mathcal{C}^{X,Y}_{\text{mutex}}$ be defined like $\mathcal{C}^{X,Y}$ in Proposition 10.6 but with the state equation heuristic's extension to mutexes. Then $\mathcal{C}^{X,Y}_{\text{mutex}}$ is $c^{\text{flow}}_{s,\alpha}$ for the constrained projection to $\{X, Y\}$ simplified with Rules 3, 4, and 5.*

Ignoring a Single Abstract State

In the flow constraint of any transition system TS, the flow balance equation for a single state can be removed without affecting the set of solutions. In contrast to previous rules, where states are removed from the transition system including all of their adjacent transitions, here we only consider removing one of the flow balance equations. This is a minor modification of the LP, and the simplification is not likely to result in a performance boost in practice. However, on the theoretical side it allows us to ignore certain parts of an abstraction, which is useful for the analysis of the final extension of h^{SEQ}, *partial merges*.

We show that the flow balance equation for a state d is redundant in the presence of flow balance equations for all other states. The proof can be seen as a generalization of the proof in Section 10.3 and follows a similar argument. Since every transition has to start and end in some state we have $\bigcup_{s\in\mathcal{S}} in_{\text{TS}}(s) = \bigcup_{s\in\mathcal{S}} out_{\text{TS}}(s)$. Thus, the sum over the left-hand side of all flow balance equations except the one for d is:

$$\begin{aligned}&\sum_{s\in\mathcal{S}\setminus\{d\}} \sum_{t\in in_{\text{TS}}(s)} \mathsf{Count}_t \;-\; \sum_{s\in\mathcal{S}\setminus\{d\}} \sum_{t\in out_{\text{TS}}(s)} \mathsf{Count}_t \\ =& -\sum_{t\in in_{\text{TS}}(d)} \mathsf{Count}_t \;+\; \sum_{t\in out_{\text{TS}}(d)} \mathsf{Count}_t\end{aligned}$$

Since flow balance is expressed with equations, the sum over their left-hand sides has

to equal the sum over their right-hand sides, which is:

$$\sum_{s \in \mathcal{S} \setminus \{d\}} [s = s_\star] - \sum_{s \in \mathcal{S} \setminus \{d\}} [s = s_\mathrm{I}] = -[d = s_\star] + [d = s_\mathrm{I}]$$

Multiplying both sides by -1 results in the flow balance equation for d.

Rule 6. *Removing the flow balance equation for a single state from $c^{\mathrm{flow}}_{s,\alpha}$ does not influence the set of solutions.*

We use this rule to analyze partial merges of variables X and Y, where only the flow balance equations for a subset of values $M \subseteq \mathit{dom}(X) \times \mathit{dom}(Y)$ are part of the model.

Proposition 10.8. *Let $\alpha^{X,Y}_M$ be the abstraction with abstract states $M \cup \{d\}$ that maps every state s to $z = \langle s[X], s[Y] \rangle$ if $z \in M$ and to d otherwise. Let $\mathcal{C}^{X,Y}_M$ be defined like $\mathcal{C}^{X,Y}$ in Proposition 10.6 but only considering merged values in M. Then $\mathcal{C}^{X,Y}_M$ is the constraint that results from simplifying $c^{\mathrm{flow}}_{s,\alpha^{X,Y}_M}$ with Rules 3 and 4, and then using Rule 6 to remove the flow balance equation for d.*

10.4.2. Strengthening the State Equation Heuristic

Proposition 10.8 suggests a clean way for generalizing partial merges beyond two variables: project the task to all involved variables, then abstract the projection further by mapping all unrepresented abstract states to a new state d. The partial merge then is the flow constraint for the resulting abstraction simplified with Rules 3 and 4, then using Rule 6 to remove the constraint for d. Using Rules 1, 2, and 5 as well can strengthen the model and reduce its size. Applying all rules results in the following improved constraint:

Definition 10.1 (improved flow constraint)**.** *Let $P \subseteq \mathcal{V}$ be a set of variables and M a set of partial states over P, i.e. $\mathit{vars}(m) = P$ for $m \in M$. Let α^P_M be the abstraction with abstract states $M \cup \{d\}$ that maps every state s to $s|_P$ if $s|_P \in M$ and to d otherwise. Let* TS *be the abstract transition system of α^P_M with transitions adjacent to dead and mutex violating states removed. The* improved flow constraint *for M, $c^{\mathrm{flow}+}_{s,M}$ consists of an improved flow balance equations and linking constraints.*

For every $m \in M$ it contains the improved flow balance equation

$$\sum_{t \in \mathit{in}_{\mathrm{TS}}(m)} \mathit{var}(t) \;\; - \sum_{t \in \mathit{out}_{\mathrm{TS}}(m)} \mathit{var}(t) = [m = s_\star^{\alpha^P_M}] - [m = \alpha^P_M(s)]$$

where $\mathit{var}(t) = \mathsf{Count}_o$ if $|\mathit{trans}_{\mathrm{TS}}(o)| = 1$ and Count_t otherwise.

For each operator $o \in \mathcal{O}$ that has $|\mathit{trans}_{\mathrm{TS}}(o)| \neq 1$ and induces no self-loops in TS *it contains a strong linking constraint*

$$\sum_{t \in \mathit{trans}_{\mathrm{TS}}(o)} \mathsf{Count}_t = \mathsf{Count}_o.$$

For each operator $o \in \mathcal{O}$ that induces self-loops and state changing transitions in TS *it contains a weak linking constraint*

$$\sum_{\substack{t \in trans_{\mathrm{TS}}(o) \\ t \text{ is no self-loop}}} \mathsf{Count}_t \leq \mathsf{Count}_o.$$

An operator-counting LP heuristic over several improved flow constraints computes the optimal general operator cost partitioning over the abstraction heuristics for the abstractions α_M^P. If we use improved flow constraints with $M = dom(V)$ for all variables V, the resulting heuristic dominates the state equation heuristic. If we add improved flow constraints for larger partial states, the resulting heuristic dominates the state equation heuristic where these partial states are represented as a partially merged value. In both cases the model has the same size as that of the state equation heuristic and the dominance is strict because of the strengthened linking constraints and removed dead states.

10.5. Limits of Operator Counting

Every heuristic can be written as an operator-counting heuristic with the trivial constraint $\sum_{o \in \mathcal{O}} \mathsf{Count}_o cost(o) \geq h(s)$. However, this constraint does not add any additional information besides the heuristic value of h, i.e. the operator-counting LP heuristic over several such constraints only computes their maximum. To take full advantage of the operator-counting framework, we are interested in heuristics that can be represented with operator-counting constraints and are independent of the operator cost. Here, we show that not all heuristics can be represented this way.

Definition 10.2 (expressible as operator-counting heuristics)**.** *Let Π be a planning task and s one of its states. A function h that maps states and cost functions of Π to heuristic estimates $h(s, cost) \in \mathbb{R} \cup \{-\infty, \infty\}$* can be expressed with an LP operator-counting heuristic in *s if there is an operator-counting constraint C for s such that $h^{\mathrm{LP}}_{\{C\}}(cost) = h(s, cost)$ holds for all cost functions. It* can be expressed with a MIP operator-counting heuristic *if $h^{\mathrm{MIP}}_{\{C\}}(cost) = h(s, cost)$ holds for all cost functions.*

The function h can be expressed as an LP/MIP operator-counting heuristic *if it can be expressed in this way in every state s.*

The heuristics discussed earlier provide examples for heuristics that can be expressed as LP and MIP operator-counting heuristics. For example, abstraction heuristics can be expressed with improved flow constraints and landmark heuristics can be expressed with landmark constraints. To show that not every heuristic can be expressed this way, we prove a necessary property of such heuristics.

Proposition 10.9. *A function h that can be expressed as an LP or MIP operator-counting heuristic in a state s is* superadditive*, i.e. for all cost functions $cost_1$ and $cost_2$:*

$$h(s, cost_1 + cost_2) \geq h(s, cost_1) + h(s, cost_2).$$

(As before, we define sums involving ∞ as ∞, even if they also include $-\infty$ but this case can not occur here.)

Proof: Let h be a function that can be expressed as an LP or MIP operator-counting heuristic in a state s with an operator-counting constraint C. Let S_C be the set of solutions of the operator-counting LP or MIP. We write the value of a solution $f \in S_C$ under cost function *cost* as $val(f, cost) = \sum_{o \in \mathcal{O}} f(\mathsf{Count}_o) cost(o)$.

All three heuristic values are ∞ iff S_C is empty and in this case the inequality is satisfied. So assume no heuristic value is ∞.

If $h(s, cost_1) = -\infty$ or $h(s, cost_2) = -\infty$, the inequality is trivially satisfied because the right-hand side is $-\infty$.

If $h(s, cost_1 + cost_2) = -\infty$, then there have to be solutions in S_C with arbitrarily low $val(cost_1 + cost_2)$. As the value under the summed cost functions is the sum of values under the individual cost functions $val(f, cost_1 + cost_2) = val(f, cost_1) + val(f, cost_2)$, this implies that $val(f, cost_1)$ or $val(f, cost_2)$ must achieve arbitrarily low values as well. Thus, $h(s, cost_1)$ or $h(s, cost_2)$ must be $-\infty$.

In case all heuristic values are finite, we use that the minimum over a sum is at least as high as the sum over minima:

$$\begin{aligned}
h(s, cost_1 + cost_2) &= \min_{f \in S_C} val(f, cost_1 + cost_2) \\
&= \min_{f \in S_C} (val(f, cost_1) + val(f, cost_2)) \\
&\geq \min_{f \in S_C} val(f, cost_1) + \min_{f \in S_C} val(f, cost_2) \\
&= h(s, cost_1) + h(s, cost_2)
\end{aligned}$$

□

Landmark heuristics, abstraction heuristics and delete-relaxation heuristics can be expressed with operator-counting constraints and cover three out of the four common heuristic classes. The fourth class consists of critical path heuristics (Haslum and Geffner, 2000), and they have been notably absent in the discussion so far. In particular this class includes the well-known heuristic $h^{\max}$ (Bonet and Geffner, 2001). Unfortunately, they cannot be represented with operator counting.

The critical path heuristics h^m are defined for tasks in the STRIPS formalism (Fikes and Nilsson, 1971) instead of SAS$^+$ tasks. For our purposes, the difference is not important as the task used in the proof can be seen as a SAS$^+$ and as a STRIPS task. They estimate the cost to reach a set of atoms by the cost of the most expensive-to-reach subset of size m. The cost of such subsets is defined by a system of equations. If all

atoms in the set are already true in the initial state, their cost is 0. Otherwise, their cost is the cheapest way of reaching the regression of the set through an operator o plus the cost of o.

Proposition 10.10. *Critical path heuristics h^m cannot be expressed as LP or MIP operator-counting heuristics.*

Proof: Consider a task with $2m$ binary variables that all have the value 0 in the initial state and the value 1 in the goal state. For each variable, there exists one operator with cost 1 that sets the variable from 0 to 1 without additional preconditions. Let cost_1 be the functions that assigns a cost of 1 to the first m operators and a cost of 0 to the other operators. Analogously, cost_2 assigns a cost of 1 to the other m operators, so $\mathit{cost}_1 + \mathit{cost}_2$ assigns a cost of 1 to all operators.

Now we have $h^m(s_\mathrm{I}, \mathit{cost}_1) = h^m(s_\mathrm{I}, \mathit{cost}_2) = m$ but also $h^m(s_\mathrm{I}, \mathit{cost}_1 + \mathit{cost}_2) = m$ because under all three cost functions, reaching the most expensive subset of m goal conditions requires m operators with cost 1. This means that h^m is not superadditive:

$$h^m(s_\mathrm{I}, \mathit{cost}_1 + \mathit{cost}_2) = m < 2m = h^m(s_\mathrm{I}, \mathit{cost}_1) + h^m(s_\mathrm{I}, \mathit{cost}_2).$$

□

Intuitively, the necessary condition of superadditivity means that if a heuristic can benefit from cost partitioning *with itself* then it cannot be expressed as an operator-counting heuristic. We have already seen heuristics that can benefit from cost partitioning with other heuristics and can be expressed.

11. Experiments

We evaluate the different types of operator-counting techniques in two steps: first we consider constraint groups of a single type, like landmark constraints, individually. Operator counting offers an interesting way to combine different heuristics, so we next investigate combinations of such constraint groups. For these experiments, we define the following following groups for the different types of operator-counting constraints:

LMC Landmark constraints for all landmarks found by the LM-cut heuristic.

DEL, DEL$_{\mathsf{TR}}$ The delete-relaxation constraint and its time relaxation. The heuristic $h^{\text{MIP}}_{\text{DEL}}$ is the delete-relaxation heuristic h^+.

PHO$_k$ PDB constraints for all interesting patterns with up to k variables. The heuristic $h^{\text{LP}}_{\text{PHO}_k}$ is the post-hoc optimization heuristic $h^{\text{pho}}_{\text{Int}_k}$.

SEQ Lower bound net change constraints for all atoms. The heuristic $h^{\text{LP}}_{\text{SEQ}}$ is the state equation heuristic h^{SEQ}. We also consider extending this set with prevail constraints and upper bound net change constraints.

FLOW$_k$,FLOW^+_k All (improved) flow constraints for projections to patterns of up to k variables (All$_k$). Improved flow constraints are flow constraints simplified with all our simplification rules. We also evaluate using only some of the rules to measure their effect. The heuristic $h^{\text{LP}}_{\text{FLOW}_1}$ computes an optimal general operator cost partitioning over atomic flow heuristics, and $h^{\text{LP}}_{\text{FLOW}^+_1}$ computes such a cost partitioning over constrained atomic projections.

11.1. Individual Constraint Groups

To get an idea of the quality of the different constraint groups, we ran an A* search with operator-counting MIP and LP heuristics using one of the above configurations at a time. Experiments for flow constraints use the transition normalization of all tasks. The results for FLOW$_{\mathcal{C}}$ and FLOW$^+_{\mathcal{C}}$ are therefore not directly comparable to the other results and we discuss them separately. The resulting coverage for the remaining constraint groups is reported in Tables 11.1 and 11.2. Table 11.1 also shows the state-of-the-art heuristic $h^{\text{LM-cut}}$ for comparison.

Optimal cost partitioning on LM-cut landmarks ($h^{\text{LP}}_{\text{LMC}}$) leads to the highest coverage of operator-counting LP heuristics with a clear lead over the state equation heuristic

			h^{LP}						
Year	**Domian**	$h^{\text{LM-cut}}$	**LMC**	**SEQ**	**PHO$_1$**	**PHO$_2$**	**PHO$_3$**	**DEL**	**DEL$_{\text{TR}}$**
2004	Airport (50)	**28**	**28**	21	23	26	14	11	11
2011	Barman (20)	**4**	**4**	**4**	**4**	**4**	0	0	0
2014	Barman (14)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2000	Blocks (35)	**28**	**28**	**28**	**28**	26	19	19	19
2014	ChildSnack (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2002	Depot (22)	**7**	**7**	**7**	**7**	**7**	4	2	2
2002	DriverLog (20)	**13**	**13**	12	12	**13**	12	10	10
2008	Elevators (30)	**22**	20	9	11	18	20	3	4
2011	Elevators (20)	**18**	16	7	9	15	16	1	1
2011	FloorTile (20)	**7**	6	4	2	2	2	2	2
2014	FloorTile (20)	**6**	5	2	0	0	0	0	0
2000/2	Freecell (80)	15	15	**38**	14	14	7	6	6
2014	GED (20)	**15**	**15**	13	**15**	**15**	**15**	7	7
2014	Grid (5)	**2**	**2**	1	1	**2**	**2**	1	1
1998	Gripper (20)	**7**	**7**	**7**	**7**	**7**	6	5	5
2014	Hiking (20)	9	9	9	**11**	**11**	9	5	5
2000	Logistics (28)	20	20	16	16	**21**	**21**	10	10
1998	Logistics (35)	**6**	**6**	3	4	5	5	2	2
2000	Miconic (150)	**141**	**141**	51	50	52	50	116	119
1998	Movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
1998	Mprime (35)	**22**	**22**	20	**22**	21	19	8	8
1998	Mystery (30)	**17**	16	14	15	15	13	8	8
1998	NoMystery (20)	14	14	10	10	16	**17**	6	6
2006	Openstacks (30)	**7**	**7**	**7**	**7**	**7**	**7**	5	5
2008	Openstacks (30)	**21**	19	16	20	19	17	6	7
2011	Openstacks (20)	**16**	13	10	15	15	12	1	4
2014	Openstacks (20)	**3**	**3**	1	**3**	**3**	1	0	0
2008	ParcPrinter (30)	18	18	**28**	15	16	23	17	17
2011	ParcPrinter (20)	13	13	**20**	11	13	18	13	13
2011	Parking (20)	3	2	3	**5**	1	0	0	0
2014	Parking (20)	3	3	3	**5**	0	0	0	0
2006	Pathways (30)	**5**	**5**	4	4	4	4	4	4
2008	PegSolitaire (30)	**28**	27	**28**	27	27	26	18	20
2011	PegSolitaire (20)	**18**	17	**18**	17	17	16	5	8
2004	PipesWorld notankage (50)	**17**	**17**	15	**17**	15	11	8	8
2004	PipesWorld tankage (50)	**12**	11	11	10	9	6	5	5
2004	PSR small (50)	49	49	**50**	49	49	48	44	44
2002/4	Rovers (40)	**8**	7	6	6	7	7	4	5
2002	Satellite (36)	7	7	6	6	6	6	**8**	**8**
2008	Scanalyzer (30)	**15**	**15**	14	12	7	7	8	8
2011	Scanalyzer (20)	**12**	**12**	11	9	4	4	5	5
2008	Sokoban (30)	**29**	28	19	23	28	17	7	7
2011	Sokoban (20)	**20**	**20**	16	19	**20**	14	4	4
2006	Storage (30)	**15**	**15**	**15**	**15**	**15**	14	11	11
2014	Tetris (17)	6	5	**12**	5	3	1	1	1
2011	Tidybot (20)	**14**	**14**	6	12	13	13	1	1
2014	Tidybot (20)	**8**	**8**	0	4	6	7	0	0
2004	TPP (30)	7	6	**8**	6	6	6	5	5
2008	Transport (30)	**11**	**11**	**11**	**11**	**11**	**11**	6	6
2011	Transport (20)	6	6	6	6	6	**7**	1	2
2014	Transport (20)	**6**	**6**	4	4	4	**6**	1	1
2004	Trucks (30)	**10**	**10**	9	6	7	7	3	3
2011	VisitAll (20)	11	10	**17**	16	16	15	15	15
2014	VisitAll (20)	5	5	**13**	12	11	9	10	10
2008	WoodWorking (30)	**17**	16	13	10	15	15	7	7
2011	WoodWorking (20)	**12**	11	8	5	10	10	2	2
2002	Zenotravel (20)	**13**	**13**	9	9	11	11	8	8
	Sum (1667)	**876**	853	723	692	721	657	485	500

Table 11.1: Coverage of LP heuristics with individual constraint sets.

		h^{MIP}						
Year	**Domain**	**LMC**	**SEQ**	**PHO$_1$**	**PHO$_2$**	**PHO$_3$**	**DEL**	**DEL$_{\text{TR}}$**
2004	Airport (50)	**28**	17	20	23	13	9	11
2011	Barman (20)	**4**	0	0	0	0	0	0
2014	Barman (14)	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2000	Blocks (35)	**28**	22	25	21	17	13	17
2014	ChildSnack (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2002	Depot (22)	**7**	4	4	5	2	2	2
2002	DriverLog (20)	**13**	9	10	10	10	4	7
2008	Elevators (30)	**17**	3	9	8	9	0	0
2011	Elevators (20)	**14**	1	7	6	7	0	0
2011	FloorTile (20)	**6**	2	0	0	0	0	0
2014	FloorTile (20)	**3**	0	0	0	0	0	0
2000/2	Freecell (80)	12	**27**	9	8	6	2	2
2014	GED (20)	**13**	7	**13**	**13**	9	3	5
2014	Grid (5)	**2**	1	1	1	**2**	0	0
1998	Gripper (20)	**6**	5	**6**	**6**	5	2	4
2014	Hiking (20)	**8**	7	**8**	7	7	0	4
2000	Logistics (28)	**20**	12	14	**20**	**20**	10	10
1998	Logistics (35)	**6**	2	2	4	4	2	2
2000	Miconic (150)	**140**	40	45	43	40	50	77
1998	Movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**
1998	Mprime (35)	**20**	15	18	17	14	1	4
1998	Mystery (30)	**16**	12	13	13	11	4	7
1998	NoMystery (20)	14	8	8	14	**16**	3	5
2006	Openstacks (30)	**7**	6	**7**	**7**	5	0	5
2008	Openstacks (30)	16	9	**17**	16	14	1	6
2011	Openstacks (20)	11	3	**12**	11	8	0	1
2014	Openstacks (20)	**1**	0	**1**	**1**	**1**	0	0
2008	ParcPrinter (30)	18	**28**	12	7	12	15	19
2011	ParcPrinter (20)	13	**20**	7	3	8	11	14
2011	Parking (20)	**1**	**1**	**1**	0	0	0	0
2014	Parking (20)	**2**	0	**2**	0	0	0	0
2006	Pathways (30)	**5**	4	4	4	4	4	4
2008	PegSolitaire (30)	26	26	**27**	24	18	2	10
2011	PegSolitaire (20)	16	16	**17**	14	7	0	1
2004	PipesWorld notankage (50)	**16**	11	13	11	8	1	6
2004	PipesWorld tankage (50)	**9**	7	6	6	4	1	4
2004	PSR small (50)	48	**49**	48	47	46	40	41
2002/4	Rovers (40)	**7**	4	4	6	6	4	4
2002	Satellite (36)	**7**	4	5	4	4	4	6
2008	Scanalyzer (30)	**13**	9	7	9	7	4	5
2011	Scanalyzer (20)	**10**	6	4	6	4	1	2
2008	Sokoban (30)	**27**	9	16	22	12	2	5
2011	Sokoban (20)	**20**	7	13	19	9	0	2
2006	Storage (30)	**15**	12	14	14	11	7	7
2014	Tetris (17)	4	**11**	3	1	0	0	0
2011	Tidybot (20)	**13**	3	5	6	7	0	1
2014	Tidybot (20)	**7**	0	0	0	0	0	0
2004	TPP (30)	**6**	**6**	5	5	**6**	5	5
2008	Transport (30)	**11**	7	9	8	10	2	5
2011	Transport (20)	**6**	2	4	3	5	0	0
2014	Transport (20)	**6**	1	4	1	3	0	1
2004	Trucks (30)	**9**	6	3	5	5	1	2
2011	VisitAll (20)	10	**16**	**16**	15	15	8	15
2014	VisitAll (20)	5	**12**	10	10	9	1	9
2008	WoodWorking (30)	**16**	10	7	13	11	7	10
2011	WoodWorking (20)	**11**	6	2	8	6	2	5
2002	Zenotravel (20)	**11**	8	8	9	8	6	6
	Sum (1667)	**810**	543	555	564	495	264	388

Table 11.2: Coverage of MIP heuristics with individual constraint sets.

and the post-hoc optimization heuristics. However, the heuristics have strengths and weaknesses in different domains: using $h^{\text{LP}}_{\text{LMC}}$ instead of $h^{\text{LP}}_{\text{SEQ}}$ solves more tasks in 32 domains while the opposite is true in 11 domains. The picture is similar but less pronounced for the post-hoc optimization heuristics. For example, there are 6 domains where $h^{\text{LP}}_{\text{PHO}_2}$ solves more tasks than $h^{\text{LP}}_{\text{LMC}}$ and 28 domains where it solves fewer. Even the two delete-relaxation configurations are better in 3 domains (two VisitAll domains and Satellite) while they are worse in 50. We now discuss the results in more detail for each of the constraint types.

11.1.1. Landmarks

The standard LM-cut heuristic computes a greedy cost partitioning over the discovered landmarks (Helmert and Domshlak, 2009). Comparing its coverage to the LP heuristic for LMC reveals that the additional effort of computing the *optimal* cost partitioning for the same landmarks does not pay off in terms of providing sufficiently better guidance. The initial heuristic value is only increased by cost partitioning in 7 domains. In 5 additional domains the number of expansions needed to reach the last f-layer decreases slightly because other heuristic values improved. However, in none of the domains does the increase in heuristic quality speed up the search enough to solve more tasks. Because of the additional overhead of solving an LP for every state, coverage decreases by 1 in 14 domains and by up to 3 in 4 domains. A possible reason for this is that the LM-cut heuristic already approximates h^+ very closely, and the corresponding LP heuristic is also bounded by h^+.

Computing the MIP instead of the LP over LMC only rarely raises the heuristic estimate. The initial state has a higher estimate with $h^{\text{MIP}}_{\text{LMC}}$ in one task of the domain Elevator and in 30 tasks of the three Transport domains. The number of expansions necessary to reach the last f-layer is only reduced in these domains and in the domain FloorTile. The number of expansions is never reduced enough to justify the additional overhead of computing a MIP instead of an LP. In fact, the geometric mean of runtimes roughly increases by a factor of 3.

11.1.2. Delete Relaxation

The MIP heuristic $h^{\text{MIP}}_{\text{DEL}}$ is the delete-relaxation heuristic h^+, while $h^{\text{MIP}}_{\text{DEL}_{\text{TR}}}$, $h^{\text{LP}}_{\text{DEL}}$, and $h^{\text{LP}}_{\text{DEL}_{\text{TR}}}$ relax the restriction of time steps, the restriction to integer variables, or both. All versions are expensive to compute and solve significantly fewer tasks than the other operator-counting heuristics.

The effect of the time relaxation is different for the LP and the MIP. In the LP heuristic removing the time steps has very little effect on the initial heuristic values. The heuristic value is only reduced by 1 in 43 tasks from 5 domains. All other initial heuristic values are unchanged. In MIP heuristics on the other hand, time relaxation can significantly reduce heuristic value as shown in the left plot of Figure 11.1.

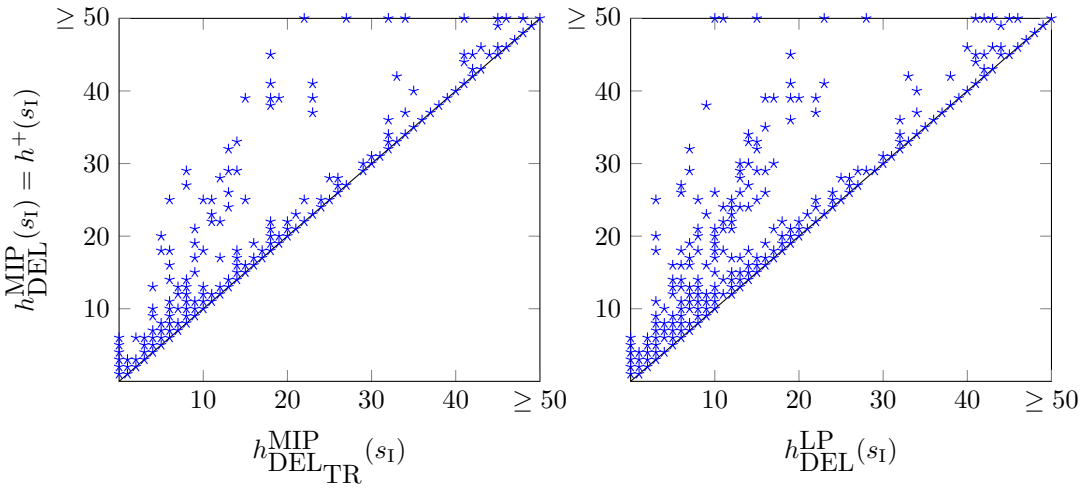


Figure 11.1: Heuristic values of the initial state for the delete-relaxation heuristic (y-axis), its time relaxation (left), and its LP relaxation (right).

The LP relaxation also significantly reduces the heuristic quality of h^+, as shown in the right plot of Figure 11.1. Compared to this loss, the LP heuristic for LMC loses much less information when we see it as a relaxation of h^+. Figure 11.2 compares the heuristic values of $h^{\text{LP}}_{\text{LMC}}$ with its theoretical optimum h^+ and with the relaxation $h^{\text{LP}}_{\text{DEL}_{\text{TR}}}$. Optimal cost partitioning of LM-cut landmarks leads to higher heuristic values than $h^{\text{LP}}_{\text{DEL}_{\text{TR}}}$ in a vast majority of cases. Combined with the fact that $h^{\text{LP}}_{\text{LMC}}$ only uses one constraint for each landmark while $h^{\text{LP}}_{\text{DEL}_{\text{TR}}}$ has at least $|\mathcal{O}| + |\mathcal{A}|$ constraints, this explains the poor performance of $h^{\text{LP}}_{\text{DEL}_{\text{TR}}}$.

11.1.3. Post-hoc Optimization

Post-hoc optimization constraints form a compromise between maximizing a set of admissible heuristics and computing an optimal cost partitioning for them. In the special case where the component heuristics are PDBs, post-hoc optimization also dominates the canonical heuristic. Their performance depends on the set of heuristics they combine. Here, we run them with projections to up to 1, 2, and 3 variables. Increasing the number and size of projections generally increases heuristic quality but makes the heuristic harder to compute. The collection Int_2 is the best trade-off in our case, both when computing LPs and MIPs. The coverage of $h^{\text{LP}}_{\text{PHO}_2} = h^{\text{pho}}_{\text{Int}_2}$ is almost exactly on par with that of $h^{\text{LP}}_{\text{SEQ}} = h^{\text{SEQ}}$ but has strengths in different domains.

As with the previous constraints, computing a MIP instead of an LP does not pay off for post-hoc optimization constraints even though it increases heuristic quality in some domains. The difference in quality is most notable in the domain Tetris, where initial heuristic values almost double for most tasks. The increase is much smaller in the other 8 domains where the initial heuristic values differ. But even on the Tetris tasks

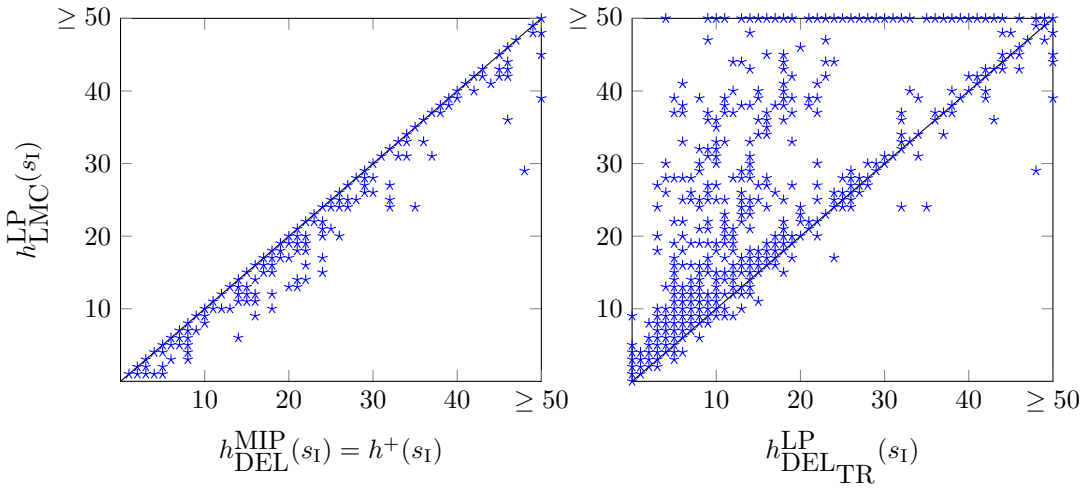


Figure 11.2: Heuristic values of the initial state for cost-partitioned LM-cut landmarks (y-axis), the delete-relaxation heuristic (left), and its time-relaxed LP-relaxation (right).

the overhead in runtime is too high and coverage drops when switching from LP to MIP heuristics.

Table 11.3 repeats the coverage obtained with different cost partitioning methods from Part I. Post-hoc optimization is shown between the canonical heuristic which it dominates and non-negative operator cost partitioning which dominates it. The canonical heuristic performs best for projections to goal variables ($k = 1$) but falls behind the other two methods for larger patterns. Post-hoc optimization solves 63 tasks more than maximization on these patterns but 25 fewer than the canonical heuristic. Post-hoc optimization performs better than both maximization and the canonical heuristic for $k = 2$. While the behavior is not consistent over *all* domains, the post-hoc optimization heuristic solves significantly more instances in many domains, overall outperforming the canonical heuristic by 65 and maximization by 54 tasks. For even larger patterns ($k = 3$), both the canonical heuristic and the post-hoc optimization heuristic are less effective. Maximization, on the other hand, improves coverage by 29 solved tasks when changing from $k = 2$ and $k = 3$. The performance of post-hoc optimization for Int_3 is between those of the other two methods again. Overall, post-hoc optimization for Int_2 achieves the highest performance of these methods to combine PDB heuristics.

To understand the difference between the canonical heuristic and post-hoc optimization, Figure 11.3 compares their expansions, evaluation rate and setup time. There are only a few instances where the theoretical dominance of the post-hoc optimization heuristic translates into better guidance. The difference in coverage thus must be due to the time spent on the two heuristics. The canonical heuristic only evaluates a maximum over sums instead of solving an LP in each state so its evaluation rate typically is orders of magnitude faster. However, the set of considered subsets that the heuristic maximizes

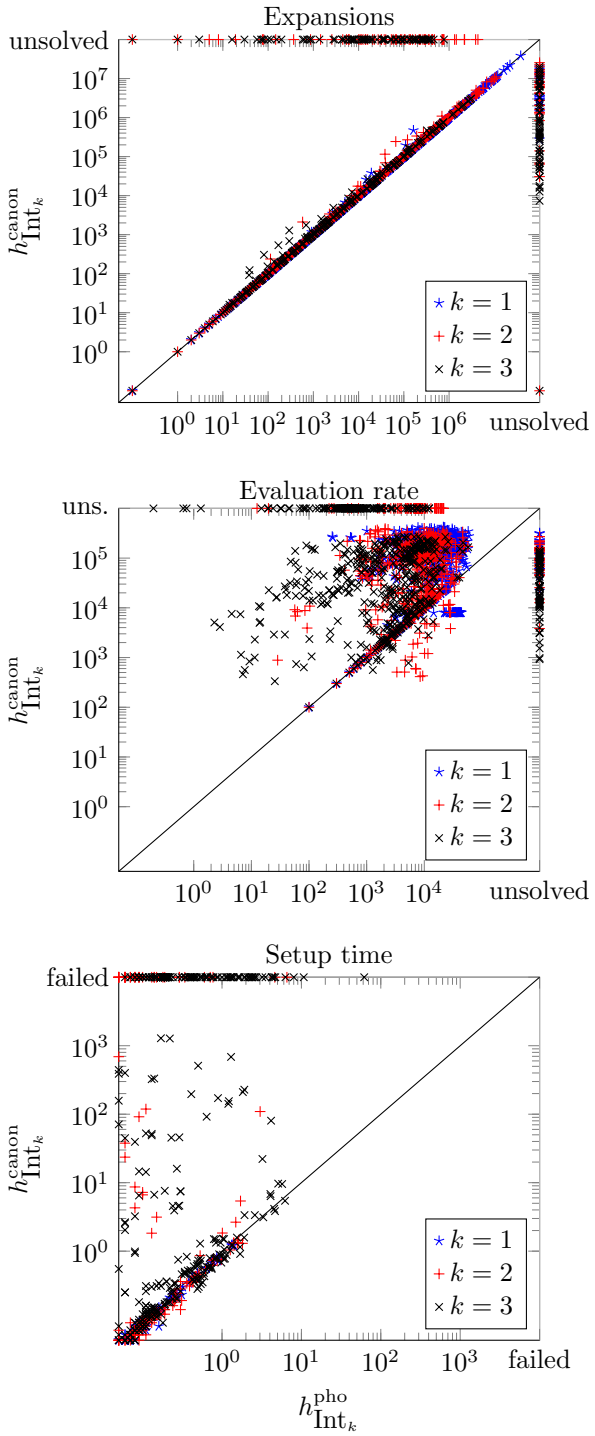


Figure 11.3: Difference between post-hoc optimization and the canonical heuristic. We compare their number of expansions in an A* search (top); their evaluation rate in states per second (middle); and setup time in seconds (bottom).

			k		
			1	2	3
Int_k	Maximization	$\max_{P \in \text{Int}_k} h^P$	629	667	**696**
Int_k	Canonical heuristic	$h^{\text{canon}}_{\text{Int}_k}$	**717**	656	597
Int_k	Post-hoc optimization heuristic	$h^{\text{pho}}_{\text{Int}_k}$	692	**721**	657
Int_k	Non-negative operator cost partitioning	$h^{\text{OCP+}}_{\text{Int}_k}$	621	438	237
All_k	Non-negative operator cost partitioning	$h^{\text{OCP+}}_{\text{All}_k}$	560	340	171
Int_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{Int}_k}$	622	475	248
All_k	General operator cost partitioning	$h^{\text{OCP}}_{\text{All}_k}$	601	357	155
Int_k	Non-negative transition cost partitioning	$h^{\text{TCP+}}_{\text{Int}_k}$	120	76	26
All_k	General transition cost partitioning	$h^{\text{TCP}}_{\text{All}_k}$	115	87	74

Table 11.3: Number of solved tasks. Best result in each column in bold.

over can become exponentially large, so we see some cases below the diagonal where post-hoc optimization evaluates more states per second. Even though post-hoc optimization only rarely decreases the necessary number of expansions and expands fewer states per second on average, it still outperforms the canonical heuristic in many cases. The reason can be seen in the last plot of Figure 11.3: the canonical heuristic computes maximal additive subsets of the pattern collection in a preprocessing step. This involves finding all maximal cliques in the graph of patterns where two patterns are connected iff they are additive. This step is NP-equivalent and takes time exponential in the number of patterns. When only projections to goal variables are considered, the pattern collections are small, and the maximal cliques can be found quickly. However, even collections of patterns with two variables can become so big that their maximal cliques cannot be determined within the resource limits. Post-hoc optimization is a polynomial algorithm that dominates the canonical heuristic. While its per-state overhead is higher, its polynomial scaling makes it useful for pattern collections that are so large that their maximal additive subsets cannot be computed in reasonable time.

11.1.4. Net Change

The operator-counting LP heuristic for the net change constraints SEQ is the state equation heuristic h^{SEQ}. It achieves the second highest coverage after the landmark constraints.

To measure the impact of the safety-based improvement of the state equation heuristic, we conducted an additional experiment where we extended SEQ with the corresponding upper bound net change constraints. This is implemented as changing con-

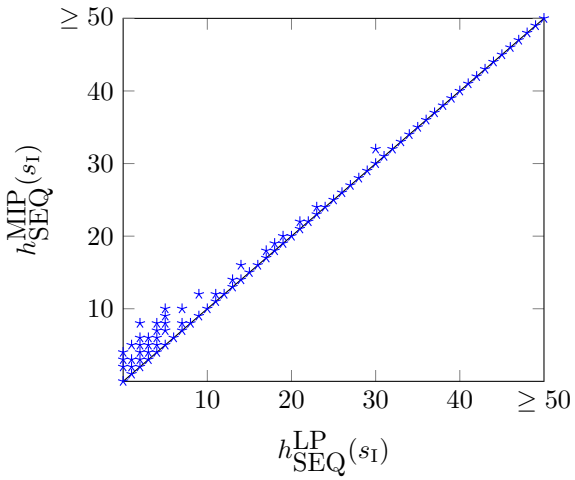


Figure 11.4: Initial heuristic values produced by operator-counting LP and MIP heuristics with lower bound net change constraints.

straints for safe variables into equations. As expected from Corollary 10.2, this has no effect on the number of expanded nodes. However, with the additional constraints we solve 30 fewer tasks across 18 domains, which can be attributed to slower evaluations of the LP solver.

We do not have a theoretical result that the upper bound net change constraints are strictly weaker than the lower bound net change constraints. However, if we solely use all upper bound net change constraints, the LP heuristic returns very poor estimates. In 26 domains the search expands more states resulting in a coverage of only 661 tasks. So these constraints are indeed strictly weaker.

We also tried adding all positive and negative prevail constraints to the set of constraints, but this never influences the heuristic value. Coverage is mostly unaffected by this and only shows differences of one or two tasks in some domains. Overall, coverage reduces by 3 when using prevail constraints; both with LPs and MIPs.

Restricting operator-counting variables to the integers has a small positive effect on heuristic quality. Figure 11.4 shows initial heuristic values for the LP and the MIP heuristic. Switching to the MIP reduces the number of expansions by roughly an order of magnitude in the two PegSolitaire domains but did not lead to more solved tasks. In fact, the LP heuristic solves 2 tasks more than the MIP heuristic in each of the two domains. There are 7 more domains where the restriction to integers reduced the number of expansions but in all of them the number of expansions stays in the same order of magnitude. Only one task in the domain Woodworking is solved with a MIP and not solved with an LP.

We have also seen that h^{SEQ} is closely related to general operator cost partitioning over atomic projections ($h^{\text{OCP}}_{\text{All}_1}$). If the abstractions contain no dead states the two heuristics are the same. The states of atomic projections are variable values, so the heuristics

	Non-negative costs	General costs
Singleton goal patterns	$h^{\text{OCP+}}_{\text{Int}_1}$: 621	$h^{\text{OCP}}_{\text{Int}_1}$: 622
All singleton patterns	$h^{\text{OCP+}}_{\text{All}_1}$: 560	$h^{\text{OCP}}_{\text{All}_1}$: 601 h^{SEQ}: 723

Table 11.4: Coverage for different variants of optimal operator cost partitioning for atomic projections.

are the same iff no variable contains values that are unreachable or from which the goal value cannot be reached in the projection. This is the case for most planning tasks in our benchmark set. Though there are some exceptions: in the domain Airport planes navigate on runways, can take off and land. Planes that take off are assumed to leave for another airport, so they cannot land again. If a plane is initially on the ground and its goal is to move to another position on the ground, then taking off sets the position of the plane to "in the air", and the goal value of this variable can no longer be reached. Another example is in the domain Parking that models cars maneuvering in a car park. Due to a modeling error it is possible for a car to park behind itself, which means it then blocks itself and can never move again. In both of these cases a variable domain contains a dead value and h^{SEQ} is not guaranteed to be the same as $h^{\text{OCP}}_{\text{All}_1}$. This situation also occurs in the domains ParcPrinter, TPP, and one task in the domain Trucks. In all cases h^{SEQ} and $h^{\text{OCP}}_{\text{All}_1}$ use the same number of expansions, so the dead values never influence the heuristic value on our benchmarks.

Table 11.4 compares the coverage of h^{SEQ} to that of other methods to combine atomic projections. We have already seen in Part I that using projections to non-goal variables slows down non-negative cost partitioning without improving the heuristic value, and that this loss can be recovered to some degree by allowing negative costs. Just using general cost functions was not enough to improve coverage above that of $h^{\text{OCP+}}_{\text{Int}_1}$ or $h^{\text{OCP}}_{\text{Int}_1}$. The state equation heuristic exceeds the coverage of both by over 100 tasks. It has the heuristic accuracy of $h^{\text{OCP}}_{\text{All}_1}$ but it can be computed much faster. Due to the smaller LP size h^{SEQ} solves more tasks than $h^{\text{OCP}}_{\text{All}_1}$ in 38 domains, an overall coverage increase of 122. This shows that the main reason for h^{SEQ}'s superior performance is the more compact representation, though the general cost partitioning also plays an important role, as the comparison of $h^{\text{OCP+}}_{\text{All}_1}$ and $h^{\text{OCP}}_{\text{All}_1}$ shows.

11.1.5. Network Flow

The state equation heuristic h^{SEQ} and its extensions can be thought of as certain cost-partitioned flow heuristics. In Section 10.4 we introduced several rules that improve a naive implementation of cost-partitioned flow heuristics and used them to show their

Rule	Description	Reduces model size	Can improve heuristic
1	Remove dead abstract states.	✓	✓
2	Use operator-counting variables instead of transition-counting variables where possible.	✓	
3	Remove linking constraints for operators inducing only self-loops.	✓	
4	Use weak instead of strong linking constraints where possible.	(✓)	
5	Remove mutex-violating abstract states.	✓	✓
6	Remove constraint for a single abstract state.	(✓)	

Table 11.5: Overview of simplification rules introduced in Section 10.4. Rule 4 cannot reduce the model size for projections of TNF tasks, and Rule 6 can only reduce the size of the model by a negligible amount.

relation to extensions of the state equation heuristic. Table 11.5 shows an overview of the rules and how they influence the heuristic model. The flow constraints in FLOW_k are not simplified at all, while the improved flow constraints in FLOW_k^+ are simplified with all rules. The rules assume TNF tasks, so we use the transition normalization of all tasks for this experiment. This slightly reduces the coverage of h^{SEQ} by 5 tasks to 718.

Table 11.6 lists the coverage obtained using FLOW_k and FLOW_k^+. We observe that the performance for $k = 1$ is far worse than that of $h^{\text{OCP}}_{\text{All}_1}$ when no simplifications are used and almost reaches the performance of h^{SEQ} if all rules are used. The remaining difference to h^{SEQ} is caused by the overhead of generating the model. While using all simplification rules does not lead to a heuristic stronger than h^{SEQ}, the rules allow us to explain the advantage that h^{SEQ} has over $h^{\text{OCP}}_{\text{All}_1}$ in more detail. To do that, we now evaluate the effect of all individual rules, except for Rules 4 and 6.

The conditions of Rule 4 never apply in our benchmark. The rule only simplifies the model if an operator induces both self-loops and state-changing transitions. This can happen in general abstractions of TNF tasks and in projections of unrestricted SAS^+ tasks but not in projections of TNF tasks. A TNF operator either induces only state-changing transitions or only self-loops in a projection.

Rule 6 has almost no effect on the model and was just introduced as a theoretical tool to show the connection of partial merges to cost-partitioned abstractions.

For each other rule r and each collection size k, Table 11.6 lists the change in coverage between using unchanged flow constraints (FLOW_k) and using flow constraints improved by rule r. We also measure the change in coverage between a version im-

	$h^{\mathbf{LP}}$	
	FLOW_k	FLOW_k^+
All_1	511	716
All_2	304	377
All_3	142	149

	Effect of rule			
	1	2	3	5
All_1	$-6/3$	13/83	120/191	$-2/\pm 0$
All_2	17/21	1/7	43/53	1/1
All_3	4/4	$\pm 0/1$	6/3	$-1/1$

Table 11.6: Total coverage when using no/all simplification rules (left) and effect on coverage of different simplification rules (right). An entry of x/y for rule r represents that coverage increased by x when using r over using no simplification rules, and that coverage increased by y going from using all rules except r to using all rules.

proved with all five rules (FLOW_k^+) and one improved with all rules except r.

Dead States As mentioned before, there are only few domains where the atomic projections contain dead states and in these domains removing the dead states never increases heuristic values. In the largest task of the domain Airport the atomic projections contain a total of 142 dead abstract states. For all other domains this number is much lower. The effect of this rule on atomic projections is therefore minimal. The overhead of identifying and removing dead states slightly reduces coverage when this is the only rule used. For larger projections, the effect is more noticeable because there are more dead states to remove. In all generated models for All_1 only 0.5% of all states are dead. This number increases to 1.9% for All_2 and to 6.8% for All_3. With the exception of the domains Blocksworld, FloorTile, Movie, and Scanalyzer, all domains have dead states in the projections to 2 variables in TNF.[1] However, removing the dead states never leads to an increased heuristic value on our benchmarks. The increase in performance is solely based on faster LP evaluations.

The unsimplified flow constraints can become quite large and exceed memory limits. For example, 10 tasks in the domain Airport run out of memory while creating FLOW_1. After applying Rules 2 and 3 the constraint is small enough to be represented. Removing dead states afterwards is sufficient to increase coverage by 2 in this domain.

Operators Inducing a Single Transition In atomic projections of TNF tasks all operators induce either a single transition (if the operator mentions the variable) or self-loops on all abstract states (if it does not). As expected, Rule 2 thus has the largest effect for atomic projections. In projections to at least one variable mentioned by the operator and one not mentioned by it, an operator induces more than one transition.

[1] Note that transition normalization can introduce dead states even if the original domain has a strongly connected state space. For example, forgetting the position of a truck in the domain Logistics leads to a dead state if that truck is still needed to deliver a package.

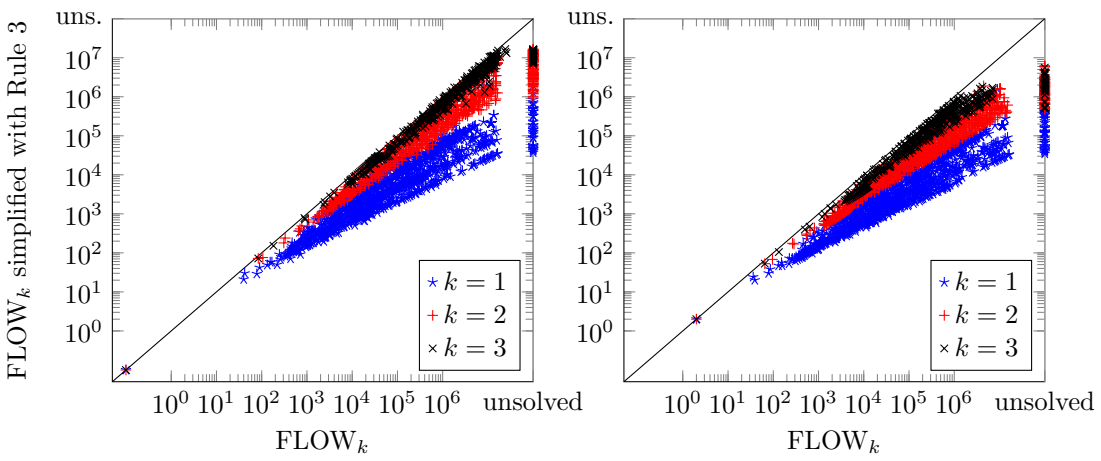


Figure 11.5: Number of LP variables (left) and constraints (right) of flow constraints FLOW$_k$ before and after applying Rule 3 to ignore operators that only induce self-loops.

The rule triggers in smaller percentage of the abstractions of larger collections and thus its effect is reduced there. The effect is significantly stronger if the other rules are used as well. We will discuss the reason for this after considering self-loops.

Operators Inducing Only Self-Loops Rule 3 triggers for operators that are not relevant for a projection because they mention none of the variables. Removing constraints and transition-counting variables for transitions of such operators drastically decreases the model size as shown in Figure 11.5. In some cases the size of the model is reduced by several orders of magnitude which makes the heuristic much faster to compute. The proportion of irrelevant operators decreases as the patterns become larger, so the effect is smaller in those cases.

The operator-counting heuristic $h^{\text{LP}}_{\text{FLOW}_k}$ computes $h^{\text{OCP}}_{\text{All}_k}$ if the abstractions contain no dead states. We have seen that dead states have little influence on the heuristic but the performance of $h^{\text{OCP}}_{\text{All}_k}$ is still far better than that of $h^{\text{LP}}_{\text{FLOW}_k}$. The coverage of $h^{\text{OCP}}_{\text{All}_k}$ is slightly reduced by transforming the tasks to TNF compared to the results from part I: on the transition normalization, $h^{\text{OCP}}_{\text{All}_k}$ solves 602, 345, and 132 tasks for $k \in \{1, 2, 3\}$. However, this does not explain the difference to flow constraints. In the dual view of cost partitioning a self-loop introduces the constraint that the cost of its operator in this abstraction cannot be negative. Our implementation handles such constraints by changing the bounds of the LP variable instead of adding an LP constraint. Therefore, our implementation of $h^{\text{OCP}}_{\text{All}_k}$ considers Rule 3 in a way. Indeed, the coverage achieved with $h^{\text{OCP}}_{\text{All}_k}$ is much closer to that achieved with flow constraints simplified with Rule 3 (631, 347, and 148).

The effect of using Rule 3 is also stronger in the presence of the other rules, as

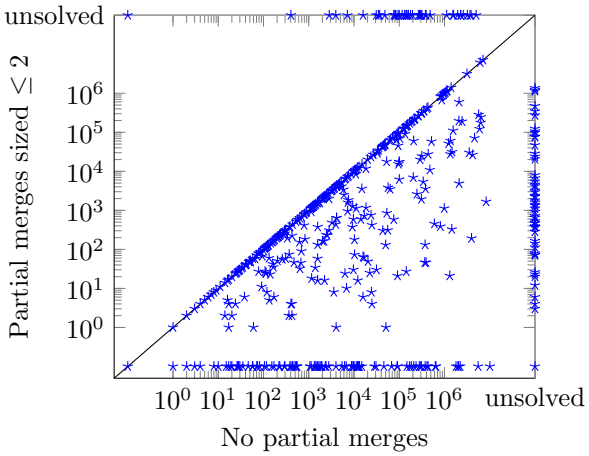


Figure 11.6: Number of expansions excluding the last f-layer for $h^{\text{LP}}_{\text{FLOW}_1^+}$ with and without partial merges of size 2.

we have seen for Rule 2. To analyze this effect, we computed geometric means over the percentage of LP constraints removed by different rules in the models of FLOW_1, excluding the 43 problems where the naive models could not be represented. Rule 2 removed 5.03% of the constraints and Rule 3 removed 90.70%. The picture is similar for LP variables (4.89%, 88.23%). Removing 5% of the original variables and constraints alone does not have a big effect, but if 90% are already removed, it removes half of the remaining constraints. The other rules amplify this effect.

Mutex Information An abstract state can only violate a mutex if it contains two mutually exclusive atoms, so Rule 5 cannot have an effect for atomic projections. Mutex information is used by Fast Downward's translator to group atoms into finite domain variables (Helmert, 2009). Additional mutex information is discovered in 24 domains. Removing abstract states that violate this mutex information can increase heuristic values if there is a "shortcut" over such a state in an abstraction. Unfortunately, this seems to be rare with our benchmarks: the initial heuristic value is only increased for FLOW_2 in 27 tasks of the Blocksworld domain and in 11 tasks across 4 other domains. None of the 11 tasks is solved and in Blocksworld the coverage only increases by 1 with a modest decrease in expanded states. The situation is similar for FLOW_3.

Partial Merges We implemented the partial merge strategy suggested by Bonet and van den Briel (2014) for the state equation heuristic starting from improved flow constraints for all atomic projections (FLOW_1^+). This basic heuristic dominates h^{SEQ} but as the previous sections showed, there is no difference on our benchmarks. The strategy then iteratively adds abstract states of larger projections. In each iteration, the current model is solved and all operators that occur with non-zero count in the solution

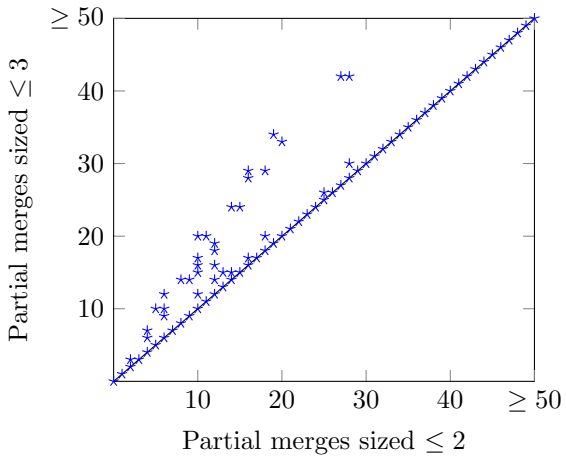


Figure 11.7: Heuristic values for the initial state for $h^{\text{LP}}_{\text{FLOW}^+_1}$ with partial merges of size 2 and 3.

are collected. For every such operator, we partition the set of preconditions into the prevail conditions and remaining preconditions. For every combination of a prevail condition $\{X = x\}$ and a non-prevail precondition $\{Y = y\}$, we add abstract state $\{X = x, Y = y\}$ to the set of explicitly represented abstract states. If all such pairs are already represented in the model, the process stops. Finally, we partially merge all pairs of variables X and Y for which an abstract state should be explicitly represented and add the resulting constraints to the current model. The model is then solved again and the process is repeated with the new solution.

As Bonet and van den Briel (2014) noted, this strategy improves the quality of the basic heuristic significantly. In our experiments, coverage increased from 716 to 783 with this strategy. Figure 11.6 shows the effect on the number of expansions.

The interpretation of partial merges as flow constraints allows us to generalize the strategy and consider larger abstract states as partial merges. We therefore extend the strategy in the following way: when there are no more abstract states with 2 variables that should be considered, we start the iteration again but now consider merges of 3 variables. Once no more merges of 3 variables are added, we look for merges of 4 variables, and so on. For such larger partial merges, we consider abstract states that contain at least one prevail condition, at least one non-prevail precondition, and take the remaining atoms from all preconditions of the operator. This process is stopped when a limit on the number of variables in a partial merge is reached, or when a solution is found where all combinations of preconditions are already represented.

Figure 11.7 compares initial heuristic values for this partial merge strategy when limited to abstract states of size 2 and 3. With larger merges much higher heuristic values can be reached but this occurs mostly in three domains (Depot, Hiking, and Storage). The number of expansions decreases in these and in nine other domains, but

the reduction in the other domains is very modest. The increased heuristic quality never translates to a better coverage and the overall coverage decreases to 726.

Interestingly, there is almost no additional benefit for limits larger than 3. Without a limit, the preprocessing runs until it exhausts resources or until the limit exceeds the number of preconditions of an operator in the solution. In this case, the search still solves 695 tasks. And while many abstract states with more than 3 variables are added in 25 domains, only two tasks in the domain Depot require fewer expansions compared to a size limit of 3.

11.2. Combination of Constraint Groups

A major advantage of operator-counting heuristics is that they are easy to combine. The operator-counting LP heuristic for constraints from several component heuristic computes their optimal cost partitioning. The MIP heuristic can combine the heuristics in an even stronger way. So far, we only looked at operator-counting heuristics for constraints of the same type. We now explore how different types of constraints interact, whether they can strengthen each other, and whether their combination is worth the additional time spent on solving a larger LP or MIP.

We restrict our attention to the three best-performing individual constraints groups: SEQ, PHO_2, and LMC. Tables 11.7 and 11.8 show the coverage of all their combinations. For a given group of constraints the column h^{LP} in Table 11.7 lists the result for combining all constraints in one LP while the column $\max$ lists results for solving one LP per constraint group and maximizing the optimal objective values. Table 11.8 has similar columns for the MIP.

A combination of SEQ and PHO_2 looks promising because they have their strengths and weaknesses in different domains. For example, using the LP heuristic for PHO_2 solves 13 tasks in the TidyBot domain of 2011, while only 6 can be solved with SEQ. In the 2011 ParcPrinter domain the picture is reversed: using SEQ, we solve 20 tasks in contrast to only 13 with PHO_2. Indeed, the combination solves 745 instances, a clear improvement on each individual heuristic solving 723 ($h^{\text{LP}}_{\text{SEQ}}$) and 721 ($h^{\text{LP}}_{\text{PHO}_2}$) tasks, respectively.

The combination of PHO_2 and LMC does not pay off, solving 852 task instead of 721 and 853, respectively. Even though the total coverage achieved by using both constraints is almost the same as using just the LMC constraint, the two heuristics solve different tasks. In 15 domains the additional overhead of using both constraints slows down the search enough so that 1–3 fewer tasks are solved. On the other hand, there are 8 domains where coverage increases due to the more accurate heuristic. For example, 13 tasks are solved with the combination in VisitAll 2014, while only 5 are solved with LMC and only 11 with PHO_2.

The best combination of two of our constraint groups consists of SEQ and LMC: with 886 task, it solves 33 more tasks than its best component, LMC, alone. This com-

bination also outperforms the standard LM-cut heuristic (with 876 tasks), which was previously the best performer among the heuristics discussed in this thesis. There are 7 domains where the combination solves more tasks than both constraints individually. For example, in the domain WoodWorking 2008, 16 tasks are solved with LMC, 13 with SEQ, and 21 with their combination.

So far, LPs for more constraints achieved at least the same level of overall coverage as their components. However, adding more constraints does not always have a positive effect. While the combination of all three components is still better than the combination of SEQ and PHO_2, searching with it solves 41 fewer instances than with the combination of SEQ and LMC.

Constraint interactions

Can we explain the better performance of the combinations with the better guidance of more individual components, or is there an additional positive effect through interactions of the different constraints in the LP? Figure 11.8 plots the number of expansions using one LP heuristic with two constraint groups against the expansions using the maximum of the two individual LP heuristics.

In all three cases, we see clear synergy effects: combining two sets of constraints in a single LP indeed leads to stronger heuristic estimates than maximizing the heuristic estimates from two separate LPs. These synergy effects are much more pronounced in the combinations of SEQ and PHO_2 and of SEQ and LMC than in the combination of PHO_2 and LMC. In all three cases, there is a solid number of tasks (10–16) that are solved with perfect heuristic estimates by the combination into one LP, but not by the maximum of two LP heuristics.

Considering coverage, however, the picture is somewhat more mixed: some tasks can only be solved by the approaches using a single large LP, others only by the maximum over two LP heuristics, and both approaches end up too close to tell apart in terms of overall coverage. Interestingly, this is different when solving the operator-counting MIP heuristic instead of the LP heuristic. Table 11.8 shows the coverage of these heuristics. In all three constraint groups, the combination in one MIP solves more tasks than the maximum over the respective MIP heuristics, but all combinations still solve fewer tasks than their strongest component.

			LMC, SEQ		LMC, PHO_2		SEQ, PHO_2		LMC, SEQ, PHO_2	
Year	**Domain**	$h^{\text{LM-cut}}$	h^{LP}	max	h^{LP}	max	h^{LP}	max	h^{LP}	max
2004	Airport (50)	28	**29**	28	27	26	26	25	25	27
2011	Barman (20)	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
2014	Barman (14)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2000	Blocks (35)	28	**29**	28	28	28	28	26	28	28
2014	ChildSnack (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2002	Depot (22)	**7**	**7**	**7**	**7**	**7**	**7**	**7**	5	**7**
2002	DriverLog (20)	**13**	**13**	**13**	**13**	**13**	12	12	**13**	**13**
2008	Elevators (30)	**22**	19	19	20	19	16	16	18	18
2011	Elevators (20)	**18**	16	16	16	16	12	13	16	15
2011	FloorTile (20)	**7**	6	6	6	6	3	3	6	6
2014	FloorTile (20)	**6**	5	5	5	4	2	1	5	4
2000/2	Freecell (80)	15	30	30	15	15	34	**35**	29	29
2014	GED (20)	**15**	12	13	**15**	**15**	11	13	10	13
2014	Grid (5)	**2**	1	**2**	**2**	**2**	**2**	**2**	1	**2**
1998	Gripper (20)	**7**	6	6	6	6	6	**7**	6	6
2014	Hiking (20)	**9**	8	8	**9**	**9**	**9**	**9**	8	8
2000	Logistics (28)	20	20	20	**21**	20	20	20	**21**	20
1998	Logistics (35)	**6**	**6**	**6**	**6**	**6**	5	5	**6**	**6**
2000	Miconic (150)	**141**	**141**	**141**	**141**	**141**	50	50	140	**141**
1998	Movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
1998	Mprime (35)	**22**	21	**22**	**22**	**22**	19	18	21	**22**
1998	Mystery (30)	**17**	16	16	16	16	13	13	15	16
1998	NoMystery (20)	14	12	14	15	**16**	14	**16**	14	15
2006	Openstacks (30)	7	7	7	7	7	**8**	7	7	7
2008	Openstacks (30)	**21**	15	15	17	17	16	16	14	15
2011	Openstacks (20)	**16**	11	11	13	12	11	11	11	10
2014	Openstacks (20)	**3**	1	1	2	1	1	1	1	1
2008	ParcPrinter (30)	18	**29**	28	19	18	**29**	28	**29**	28
2011	ParcPrinter (20)	13	**20**	**20**	14	13	**20**	**20**	**20**	**20**
2011	Parking (20)	**3**	2	1	1	1	1	1	1	1
2014	Parking (20)	**3**	**3**	**3**	0	1	1	0	0	1
2006	Pathways (30)	**5**	**5**	**5**	**5**	**5**	4	4	**5**	**5**
2008	PegSolitaire (30)	**28**	**28**	27	27	27	27	27	26	**28**
2011	PegSolitaire (20)	**18**	17	**18**	17	17	16	17	16	16
2004	PipesWorld notankage (50)	**17**	14	**17**	16	16	15	15	13	**17**
2004	PipesWorld tankage (50)	**12**	10	10	8	9	9	10	7	10
2004	PSR small (50)	49	**50**	**50**	48	49	**50**	**50**	**50**	**50**
2002/4	Rovers (40)	**8**	7	7	7	7	6	6	7	7
2002	Satellite (36)	**7**	**7**	**7**	**7**	**7**	6	6	**7**	**7**
2008	Scanalyzer (30)	**15**	12	14	13	13	12	13	11	12
2011	Scanalyzer (20)	**12**	9	11	10	10	9	10	8	10
2008	Sokoban (30)	**29**	28	28	27	28	26	27	21	27
2011	Sokoban (20)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	18	**20**
2006	Storage (30)	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
2014	Tetris (17)	6	**11**	**11**	3	3	10	**11**	6	**11**
2011	Tidybot (20)	**14**	10	13	13	13	8	7	10	13
2014	Tidybot (20)	**8**	0	7	**8**	**8**	0	0	0	7
2004	TPP (30)	7	**8**	7	6	6	**8**	**8**	**8**	7
2008	Transport (30)	**11**	**11**	**11**	**11**	**11**	10	10	**11**	**11**
2011	Transport (20)	**6**	**6**	**6**	**6**	**6**	5	5	**6**	**6**
2014	Transport (20)	**6**	**6**	**6**	**6**	**6**	4	4	5	**6**
2004	Trucks (30)	**10**	**10**	**10**	**10**	9	9	9	**10**	9
2011	VisitAll (20)	11	**19**	17	16	16	17	17	18	17
2014	VisitAll (20)	5	**15**	13	13	13	13	13	**15**	13
2008	WoodWorking (30)	17	**21**	16	18	18	15	15	**21**	17
2011	WoodWorking (20)	12	**16**	11	13	13	10	10	15	12
2002	Zenotravel (20)	**13**	12	12	12	12	11	9	12	12
	Sum (1667)	876	886	**889**	852	848	745	747	845	878

Table 11.7: Coverage of LP heuristics with multiple constraint sets.

		LMC, SEQ		LMC, PHO_2		SEQ, PHO_2		LMC, SEQ, PHO_2	
Year	**Domain**	$h^{\mathbf{MIP}}$	max	$h^{\mathbf{MIP}}$	max	$h^{\mathbf{MIP}}$	max	$h^{\mathbf{MIP}}$	max
2004	Airport (50)	**28**	**28**	25	25	19	23	24	25
2011	Barman (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2014	Barman (14)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2000	Blocks (35)	**28**	25	27	25	23	20	**28**	25
2014	ChildSnack (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2002	Depot (22)	**5**	**5**	**5**	**5**	2	4	4	**5**
2002	DriverLog (20)	11	11	**13**	12	10	10	10	10
2008	Elevators (30)	12	12	**16**	15	5	6	13	10
2011	Elevators (20)	9	10	**13**	12	4	4	10	9
2011	FloorTile (20)	**6**	4	4	2	2	2	4	2
2014	FloorTile (20)	**2**	**2**	**2**	0	0	0	**2**	0
2000/2	Freecell (80)	**24**	**24**	10	8	21	23	20	23
2014	GED (20)	7	7	**13**	**13**	7	7	7	7
2014	Grid (5)	1	1	**2**	1	1	1	1	1
1998	Gripper (20)	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
2014	Hiking (20)	6	7	**8**	7	5	7	6	7
2000	Logistics (28)	18	16	**20**	**20**	16	17	**20**	16
1998	Logistics (35)	**6**	5	**6**	**6**	2	2	**6**	5
2000	Miconic (150)	**140**	**140**	**140**	**140**	40	40	139	139
1998	Movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
1998	Mprime (35)	18	18	**20**	19	13	14	17	18
1998	Mystery (30)	13	15	**16**	15	11	11	13	14
1998	NoMystery (20)	11	12	**14**	**14**	8	10	10	**14**
2006	Openstacks (30)	6	5	**7**	**7**	**7**	5	**7**	5
2008	Openstacks (30)	9	9	**15**	14	8	7	8	8
2011	Openstacks (20)	3	3	**10**	9	2	3	2	3
2014	Openstacks (20)	0	0	**1**	**1**	0	0	0	0
2008	ParcPrinter (30)	**29**	28	19	10	**29**	17	**29**	17
2011	ParcPrinter (20)	**20**	**20**	14	5	**20**	11	**20**	11
2011	Parking (20)	**1**	**1**	**1**	**1**	0	0	0	0
2014	Parking (20)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2006	Pathways (30)	**5**	4	**5**	**5**	4	4	**5**	4
2008	PegSolitaire (30)	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**
2011	PegSolitaire (20)	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
2004	PipesWorld notankage (50)	11	11	**14**	12	11	11	11	11
2004	PipesWorld tankage (50)	**7**	**7**	6	6	**7**	**7**	**7**	**7**
2004	PSR small (50)	**50**	**50**	48	48	48	48	**50**	**50**
2002/4	Rovers (40)	**7**	**7**	**7**	**7**	6	4	**7**	**7**
2002	Satellite (36)	**6**	**6**	**6**	**6**	4	4	**6**	**6**
2008	Scanalyzer (30)	9	9	**10**	9	9	9	8	9
2011	Scanalyzer (20)	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
2008	Sokoban (30)	22	18	**23**	22	13	17	16	18
2011	Sokoban (20)	**19**	15	**19**	17	10	13	13	15
2006	Storage (30)	**14**	13	**14**	**14**	12	12	13	13
2014	Tetris (17)	**11**	**11**	2	2	7	7	4	7
2011	Tidybot (20)	6	12	**13**	**13**	4	5	6	11
2014	Tidybot (20)	0	3	**7**	**7**	0	0	0	2
2004	TPP (30)	**8**	6	6	6	6	6	7	6
2008	Transport (30)	**11**	**11**	**11**	**11**	6	6	10	**11**
2011	Transport (20)	**6**	**6**	**6**	**6**	1	2	4	**6**
2014	Transport (20)	3	3	**4**	**4**	1	1	3	3
2004	Trucks (30)	7	7	**9**	7	6	6	7	7
2011	VisitAll (20)	**18**	16	16	16	16	16	**18**	16
2014	VisitAll (20)	**15**	13	13	10	11	11	**15**	13
2008	WoodWorking (30)	**17**	14	16	15	14	12	**17**	14
2011	WoodWorking (20)	**12**	9	11	10	9	7	**12**	9
2002	Zenotravel (20)	10	10	**11**	**11**	8	8	9	10
	Sum (1667)	770	752	**781**	733	551	543	731	712

Table 11.8: Coverage of MIP heuristics with multiple constraint sets.

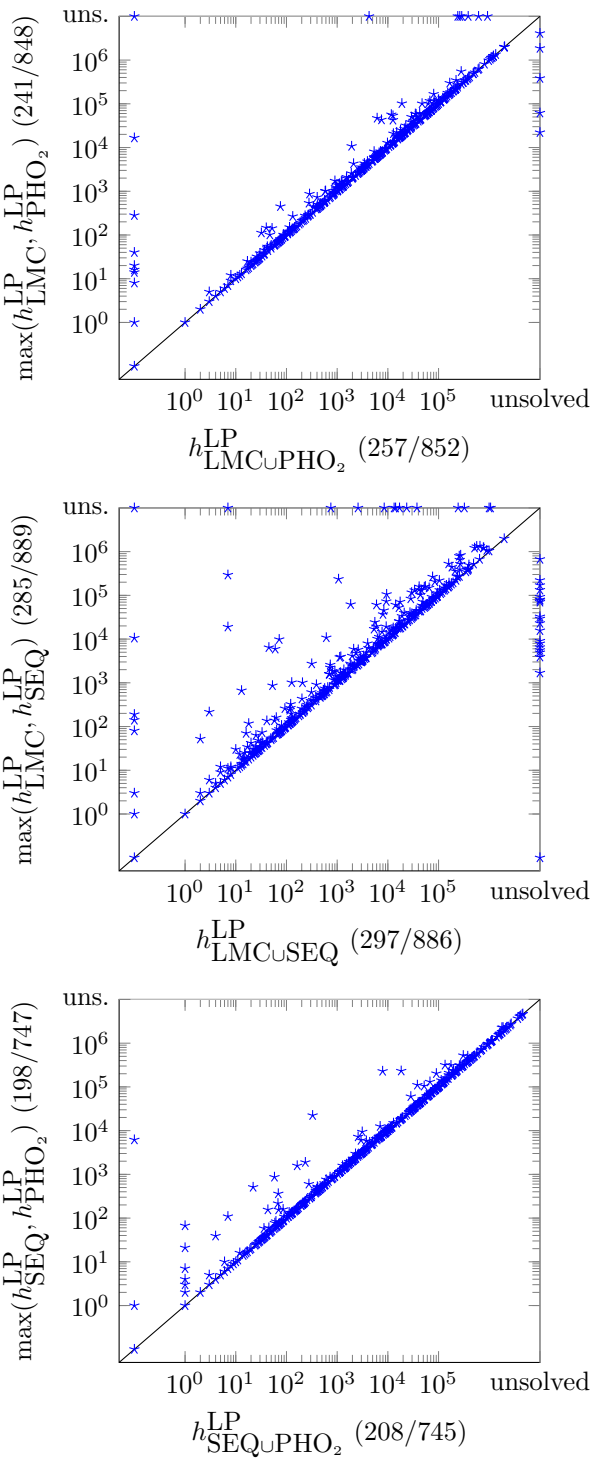


Figure 11.8: Number of expansions (excluding nodes on the final f-layer). The numbers (x/y) behind the configurations express that among the y solved tasks, x have been solved with perfect heuristic estimates.

12. Related and Future Work

In this chapter, we briefly discuss some applications of operator-counting heuristics and possible extensions of the framework to handle more general forms of planning tasks.

12.1. Under-Approximation Refinement

Heusner et al. (2014) observe that a typical planning task has significantly more operators than necessary to find a plan. Averaged over a large number of benchmarks, 4.3% of the available operators were sufficient. Restricting the set of operators of a planning task *under-approximates* the original task, i.e. every plan of the restricted task is also a plan for the original task, while the inverse is not necessarily true. Search in an under-approximation can be much faster because the branching factor and the size of the reachable state space is reduced. Heusner et al. use this for satisficing search, i.e. while searching for a plan of a planning task that does not have to be optimal. In their *under-approximation refinement* framework, the set of operators is restricted to a subset that is iteratively refined until a plan is discovered. The framework leaves open how to search in an under-approximation, how often to refine the set of operators, and how to select the operators added to the set during refinement. Operator-counting heuristics can be used for operator selection.

The solution of an operator-counting LP/MIP can be seen as an assignment of operators to the number of times they should be used. Intutively, we think about this solution as an encoded plan without its operator order. There are two problems with this view. Firstly, the solution of an LP can be fractional, in which case it does not encode a multi-set of operators. Secondly, even if the solution encodes an operator multi-set, the only guarantee about it is that its cost is a lower bound for the cost of an optimal plan. A solution could encode an inapplicable operator sequence that uses a completely different set of operators than any solution as long as the cost of these operators is admissible. As an example, consider the state equation heuristic in our example logistic task. We add an operator to "beam" the package to its destination at no cost with the prevail condition that we have a transporter (which we do not). The "beam" operator is never applicable but the state equation heuristic ignores prevail conditions and considers it an optimal solution.

But in spite of these theoretical problems, operators that have a non-negative count in a solution often appear in a plan in practice. To measure this effect, we consider an operator *predicted* if it occurs with a non-zero count in the LP solution of $h^{\text{LP}}_{\{\text{SEQ,LMC}\}}(s_{\text{I}})$.

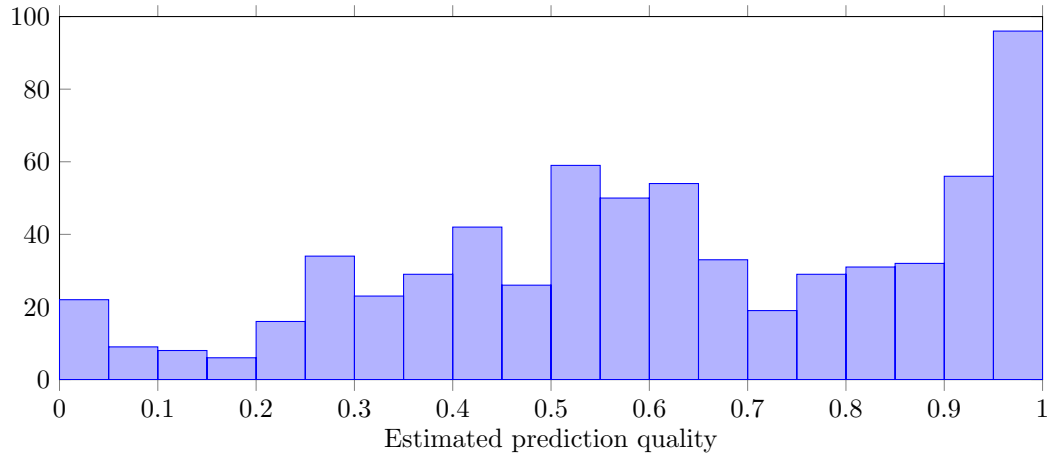


Figure 12.1: Histogram of estimated prediction quality of $h^{\text{LP}}_{\{\text{SEQ,LMC}\}}(s_{\text{I}})$. For each task, we measure length l and cost c of optimal plans for a cost function that maps each operator to 0 if it was predicted and to 1 otherwise. We use $1 - c/l$ as an estimate of prediction quality.

We find optimal plans under a cost function that assigns 0 to an operator if it was predicted and 1 if it was not. Such plans minimize the use of unpredicted operators.[1] The ratio of their length and their cost gives some indication of how good the prediction is. Figure 12.1 shows that the predicted operators often account for more than half of the operators required to find a plan. There are also 47 cases where all operators were predicted and 79 cases where all but one where predicted, but only 18 cases were no predicted operator was useful.

In an under-approximation refinement framework prediction by an operator-counting heuristic can be used to select operators to add. The operator selection method suggested by Heusner et al. uses operators from relaxed plans instead. The two approaches can be combined by adding landmark constraints for each operator in a relaxed plan. Since the operators from a relaxed plan are not guaranteed to be landmarks, these constraints are no longer operator-counting constraints and can make the LP/MIP infeasible. If the model is feasible, the predicted operators are superset of operators from the relaxed plan, which additionally can be used to satisfy all operator-counting constraints. If the model is infeasible, the given relaxed plan cannot be extended to a plan for the original problem by inserting operators. Analyzing the infeasibility could give valuable insight into the heuristic error of the relaxed plan in this situation.

This idea could also be used in connection with red-black planning (Domshlak, Hoffmann, and Katz, 2015). Red-black planning tasks partially relax the task by changing the semantics of some variables so they accumulate values (as in the delete relaxation).

[1] This may overestimate the number of necessary unpredicted operators since an unpredicted operator can be used more than once.

Operator-counting heuristics could be used to check if a given delete-relaxed plan could potentially be extended to a real plan. If this is not the case (i.e. if the LP becomes infeasible) then delete effects of operators in the given plan should be considered by changing their variables from relaxed to non-relaxed semantics.

12.2. Operator Sequencing

Davies et al. (2015) use operator-counting heuristics in a logic-based Benders decomposition (Hooker and Ottosson, 2003) for optimal planning. Logic-based Benders decomposition is problem decomposition technique that generalizes the main ideas of Benders decomposition for solving MIPs (Benders, 1962). A given problem is partitioned into a *master problem* and a number of independent *subproblems* in a way that the master problem is a relaxation of the original problem. Every solution of the master problem that is consistent with all subproblems is a solution to the original problem. The original problem can thus be solved by the following iterative process: first the master problem is solved (ignoring the subproblems). The discovered solution then fixes some parts of the subproblems (e.g. shared variables). If a subproblem is inconsistent with these values, a constraint is added to the master problem to prevent solutions that lead to this type of inconsistency. This constraint is called a *feasibility cut*. In case all subproblems can be satisfied consistently with the master solution, an *optimality cut* is added to the master problem to guarantee that only solutions with a better overall value are discovered in the next iteration. This process is repeated until the master problem is no longer solvable. Once this happens, the last consistent solution is globally optimal (if no such solution exists, the original problem is infeasible).

Davies et al. split the problem of finding optimal plans into computing an operator-counting MIP heuristic (master problem) and finding an applicable sequence of the resulting multiset of operators (subproblem). The master problem is a relaxation of the original planning task, and every master solution consistent with the subproblem is a plan. The master problem can be solved with a MIP solver and the sequencing subproblem can be encoded as a SAT problem. If the SAT problem is not satisfiable, modern SAT solvers generate a *conflict clause* that offers an explanation for the unsatisfiability. This conflict clause can be translated back into an operator-counting constraint (similar to a landmark constraint) that acts as a feasibility cut. The first consistent solution discovered this way is optimal, so no optimality cuts are necessary.

MIP solvers usually solve the LP relaxation as a first step of their search. Davies et al. speed up their approach by using the rounded-up LP solution before solving the MIP. If the subproblem is consistent and the cost of the discovered plan matches the rounded-up optimal objective value of the LP, the solution is guaranteed to be globally optimal. In case the discovered plan has a higher cost, it can be used to prune the search for a MIP solution. If there is no consistent solution for the subproblem, the generated feasibility cut excludes this LP solution in the next iteration.

12.3. Extension to Conditional Effects

In SAS$^+$ tasks, all effects of an operator are considered when it is applied in a state. More general tasks in *finite domain representation* (FDR) (Helmert, 2009) associate each effect with a partial variable assignment called the *effect condition*. When applying an FDR operator in a state s, only effects with an effect condition that is consistent with s are used.

Conditional effects can be compiled away (Nebel, 2000) but only with severe disadvantages: any plan-preserving transformation leads to an exponential blow-up of the task description size. An alternative compact compilation does not preserve the delete relaxation, which many heuristics are based on. Heuristics that handle conditional effects natively are therefore more interesting.

The operator-counting framework itself does not have to be changed to work with conditional effects. The LP and MIP heuristics are admissible, as long as all constraints are operator-counting constraints, i.e. they have a solution for every plan π where operator-counting variables are assigned to operator occurrences in π. To maintain this property in the context of conditional effects, the definitions of all operator-counting constraints introduced so far have to be generalized to cover them.

A landmark constraint for a set of operators L is an operator-counting constraint in a task with conditional effects if L is a landmark in this task. In the presence of conditional effects, an operator may be required more than once, even in the delete relaxation. Keyder, Hoffmann, and Haslum (2012) point out that obvious extensions of the LM-cut heuristic either render the heuristic inadmissible or lose the dominance over the maximum heuristic $h^{\max}$. Context splitting offers a middle ground by considering conditional effects during the landmark computation and splitting actions as necessary (Röger, Pommerening, and Helmert, 2014). The resulting context-split task can be used together with landmark constraints for the discovered landmarks in an operator-counting framework.

A post-hoc optimization constraint for a heuristic h can be extended to handle conditional effects by extending its heuristic to handle them but this can lead to further issues. For example, FDR operators can have more than one effect on the same variable, so projecting a task to a pattern can make the projection non-deterministic.

A suitably defined extension of the delete-relaxation heuristic to FDR tasks can be encoded with similar constraints as the ones introduced by Imai and Fukunaga. Instead of deciding which operator first achieves an atom, this model decides which conditional effect first adds the atom. A conditional effect can only be a first achiever if its operator is used. The time an effect can first achieve an atom must be after the first time all of the operator's preconditions and the effect conditions are achieved. All of these constraints can be encoded as linear inequalities, analogous to the model by Imai and Fukunaga.

For net change constraints, we can add variables $\mathsf{Count}_{o,e}$ for each effect e of each operator o that represent the number of times effect e triggers when applying operator o. These auxiliary variables can be connected to the operator-counting variables with

the constraint $\mathsf{Count}_o \geq \mathsf{Count}_{o,e}$, which ensures that the operator is used at least as often as the constraints trigger. If some effect conditions are mutually exclusive, this constraint can be strengthened further. Say the conditions of e_1 and e_2 are mutually exclusive, then we can add the constraint $\mathsf{Count}_o \geq \mathsf{Count}_{o,e_1} + \mathsf{Count}_{o,e_2}$. The net change induced by an operator on an atom is more complex with conditional effects, though. A normal form comparable to TNF does not exist for tasks with conditional effects and as mentioned above, an operator can induce multiple (non-deterministic) transitions in the projection to a single variable. In Appendix B.1, we show how net change constraints can be defined for tasks that are not in TNF by distinguishing operators that always produce/consume an atom from those that sometimes produce/consume it. The same technique, applied on the level of conditional effects, can be used to extend net change constraints to conditional effects.

As a side note, we remark that variables like $\mathsf{Count}_{o,e}$ may be useful in other constraints as well. While the operator-counting framework restricts the set of shared variables to operator-counting variables, this could be extended to include other variables as long as all constraints share the same interpretation of these variables. Given a plan, the value of all shared variables has to be determined and every constraint has to have a solution consistent with these values. In a similar way, the variables Time_o from delete-relaxation constraints could be shared by interpreting them as an order of the first occurrence of each operator in a given plan. Variables that count the number of times an axiom (Helmert, 2009) triggers would also be interesting for general FDR tasks, but it is less clear how the semantics of axioms could be encoded.

12.4. Extension to Other Planning Formalisms

There are many extensions of classical planning. Level 2 of PDDL 2.1 (Fox and Long, 2003) adds *numeric state variables* to the finite domain variables in level 1. Numeric planning is often combined with *temporal planning* (levels 3 and 4 of PDDL 2.1), where executing actions takes time and not all effects are instantaneous. In *probabilistic planning* the effect of an action is selected according to a probability distribution (Bellman, 1957; Bertsekas, 1995; Mausam and Kolobov, 2012). The problem of *over-subscription planning* is to find a path with a limited cost budget that maximizes a utility function over states (Smith, 2004; Aghighi and Jonsson, 2014). Heuristic search has been used in all of these formalisms (e.g. Hoffmann, 2003; Bonet and Geffner, 2003; Coles et al., 2012; Domshlak and Mirkis, 2015).

One option of coming up with admissible heuristic estimates for an extension of classical planning is to ignore the extension and compute an admissible heuristic for the "classical core" of the task. For example, this means projecting away the numeric variables (e.g. Do and Kambhampati, 2001), ignoring time (e.g. Eyerich, Mattmüller, and Röger, 2009) or transforming a non-deterministic task into a deterministic one via determinization (e.g. Yoon, Fern, and Givan, 2007; Teichteil-Königsbuch, Kuter, and

Infantes, 2010). Such heuristics ignore the aspect of the problem that distinguishes it from classical planning and are therefore usually not that informative. Heuristics that only take the distinguishing part into account (e.g. only reason about the values of numerical variables) would suffer from the same problem. Heuristics that combine both parts are often very specific to the particular planning formalism and cannot directly benefit from new ideas in classical planning.

Operator-counting heuristics might offer a general way to connect other planning formalisms with classical planning heuristics. If aspects of the problem that go beyond classical planning can be represented in terms of linear constraints and connected to the number of times an action is executed, operator-counting constraints can be combined with these constraints. Each set of constraints then focuses on one aspect of the problem but since they mention the same variables, they can interact and strengthen each other. New results for operator-counting heuristics in classical planning can then be easily used in other planning formalisms.

For example, in numeric planning, if a numeric variable is only incremented and decremented by one with operators, is 0 in the initial state and must be at least 5 in the goal, then the number of times an increment operator is used minus the number of times a decrement operator is used must be at least 5. The LPRPG heuristic (Coles et al., 2008) uses reasoning like this in an LP to derive a heuristic value. In contrast to operator-counting heuristics, their LP reasons about time steps in a relaxed planning graph (Hoffmann, 2003) but the same idea could be easily combined with operator counting. Constraints of this kind could then be combined with, say, landmarks and the state equation heuristic on the classical part.

In over-subscription planning, adding constraints that limit the cost of a plan to the given budget is straight-forward. If the utility can also be expressed in terms of operator-counting variables, it can be maximized subject to operator-counting constraints and the budget constraint to get an upper bound for the achievable utility.

One recent example of extending operator-counting heuristics to other planning formalisms is from the area of probabilistic planning. Trevizan, Thiébaux, and Haslum (2017) use *occupation measures* instead of operator-counting variables. The occupation measure of a policy π for an operator o and a state s is the expected number of times o is executed in s following policy π. They use LP variables for the occupation measures to express the expected number of times atoms are produced and consumed in atomic projections. Their heuristic is directly analogous to the state equation heuristic for classical planning. They also show how arbitrary operator-counting constraints can be translated into their framework, so any advances on classical operator-counting heuristics can be directly used for probabilistic planning.

We believe that a similar approach could work for many other planning formalisms as well and plan to work on extensions to numeric planning and over-subscription planning in the future.

13. Summary

We introduced a class of MIP/LP heuristics based on operator-counting constraints that subsumes many existing heuristics, including the state equation, post-hoc optimization and admissible landmark heuristics as well as the delete-relaxation heuristic. With an analysis of flow heuristics, we also showed that general (explicit-state) abstraction heuristics can be represented in this framework. While operator counting is very general, not every heuristic can be represented in this framework. Critical path heuristics are not superadditive (i.e. they can benefit from cost partitioning with themselves) and thus cannot be represented as operator-counting heuristics.

There are two main advantages of representing heuristics in a common language like that of operator-counting constraints:

Firstly, operator-counting constraints can be combined arbitrarily. As the main contribution in this area, we showed that operator-counting LP heuristics compute an optimal general operator cost partitioning of their component heuristics. In our experiments, the best configuration combines constraints from the state equation heuristic and from optimal cost partitioning for LM-cut landmarks. This configuration solves 13 more tasks on the IPC benchmark set than the state-of-the-art LM-cut heuristic. We saw synergy effects between most constraint types where the combination with an LP produced better heuristic estimates than the maximum. Computing the MIP heuristic for the same constraints offers an even stronger method to combine the heuristics. In practice, the combination with an LP already produces high quality heuristic estimates and the additional effort to solve an (NP-hard) MIP does not pay off in most cases.

Secondly, the relationship of different heuristics expressed in a common language is easier to analyze. For example, we show that the state equation heuristic can be seen as cost-partitioned atomic abstractions when isolated values are removed from variable domains. The proof simply compares the generated constraints, instead of having to reason about Petri nets, abstractions and shortest paths at the same time. By further analyzing the constraints, we show that extensions of the heuristic like partial merges or using mutex information correspond to other abstractions and that the safety-based extension of the state equation heuristic cannot improve heuristic accuracy. Similarly, we use the operator-counting representation to show dominance results, for example that h^{SEQ} dominates the post-hoc optimization heuristic for atomic projections if there are no dead values.

Operator counting in general makes it easy to express declarative knowledge about plans and use off-the-shelf LP solvers to combine this knowledge in an optimal way. We saw that such declarative knowledge can be extracted from a number of different

sources like landmarks or abstractions. For other planning formalism such as numeric planning, oversubscription planning, or probabilistic planning, additional information about the task could be added to encode the specific mechanics of the formalism.

Part III.

Potential Heuristics

14. Introduction to Potential Heuristics

In this part of the thesis, we introduce a new family of heuristics called *potential heuristics*. Like operator-counting heuristics, potential heuristics are *declarative*: they fix the mathematical form of the heuristic and allow us to declare properties that restrict the heuristic function. While operator-counting heuristics use properties of plans to restrict the heuristic value, potential heuristics use properties of the heuristic function itself, like consistency and admissibility.

A potential heuristic associates a numerical *weight* with a set of state features and computes the *potential* of a state as the sum of all weights of features that are present in the state. A well-known example are evaluation functions in chess (e.g. Lolli, 1763; Shannon, 1950): an approximate value of a position in chess can be computed as

$$200(K - K') + 9(Q - Q') + 5(R - R') + 3(B - B' + N - N') + (P - P')$$

where K, Q, R, B, N, and P are the number of kings, queens, rooks, bishops, knights, and pawns owned by the player, and $K', \ldots, P'$ are the number of pieces owned by the opponent. The evaluation function assigns a (positive or negative) value to each piece and includes this value in the sum if the piece is still on the board.

For now, we focus on features that can be expressed as conjunctions of atoms. More general features are conceivable (e.g. disjunctions or general formulas over atoms), and we explore a slightly broader definition in Section 16.2.

Definition 14.1 (feature)**.** *Let Π be a planning task with atoms $\mathcal{A}$. A* feature *of Π is a conjunction of atoms from $\mathcal{A}$ and the number of conjuncts is its* size*. A feature f* is true *in a state s (written as $s \models f$) if all atoms of f are true in s.*

Definition 14.2 (potential function)**.** *Let Π be a planning task and $\mathcal{F}$ a set of features of Π. A* weight function *is a function $w : \mathcal{F} \to \mathbb{R}$. The* potential function *for a weight function w maps each state s to its* potential*:*

$$\varphi^w(s) = \sum_{f \in \mathcal{F}} w(f)[s \models f].$$

The dimension *of a potential function is the maximal size of one of its features.*

We call features of size 1 and 2 *atomic* and *binary*, and likewise for potential functions of dimension 1 and 2. Potential functions can be used to estimate goal distances,

in which case we call them *potential heuristics*. They also have other uses, which we discuss in Chapter 18.

Every finite heuristic h for a task with n variables can be described as a potential function with at most n-dimensional features: in the worst case, we can define an n-dimensional feature with weight $h(s)$ for every state s. But there are interesting potential heuristics with lower dimension as well. The STRIPS heuristic (Fikes and Nilsson, 1971) counts the number of unsatisfied goals. It can be seen as an atomic potential heuristic with features $\langle V, v \rangle$ of weight 1 where V is a goal variable and v is not its goal value. The Manhattan distance for the sliding tile puzzle (Korf, 1985) sums up the Manhattan distance of every tile to its goal location and can be also be seen as an atomic potential function. In this puzzle, a *linear conflict* (Hansson, Mayer, and Yung, 1992) exists if two tiles have to pass each other in the same row or column. This can be expressed with binary potential heuristics. As a final example, note that the PDB heuristic for a pattern P is a potential heuristic of dimension $|P|$ where the features directly correspond to the abstract states in projection to P and the weights are their abstract goal distances.

15. Admissible and Consistent Potential Heuristics

The main advantage of potential functions is that their fixed mathematical structure makes it possible to analyze and reason about the heuristics. In this chapter we will show how linear constraints over feature weights can be used to express interesting properties like admissibility and consistency of potential heuristics. The constraints that we introduce *characterize* admissible and consistent potential heuristics for certain sets of features. This means that every weight function satisfying the constraints induces an admissible potential heuristic. Maximizing a formula that measures how "good" a heuristic is subject to these constraints gives the *best* admissible potential heuristic according to this criterion. If the criterion can be expressed as linear functions over the weights, an off-the-shelf LP solver can be used to automatically derive the heuristic function. In Section 15.4 we discuss different objective functions.

15.1. Atomic Potential Heuristics

We first restrict our attention to potential heuristics over feature sets with only atomic features and show that admissible and consistent atomic potential heuristics can be characterized by a compact set of linear constraints over the weights. It is again useful to restrict tasks to transition normal form (see Section 2.4) and we show in Appendix B.3 that this is not limiting generality. A generalization of the results to unrestricted SAS$^+$ is possible and the constraints for it are identical to the ones for the task's transition normalization.

Consider a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, \mathit{cost} \rangle$ in TNF and recall from Section 2.2 that a consistent heuristic is goal-aware iff it is admissible. A potential heuristic h^w for features $\mathcal{F}$ is goal-aware if and only if

$$h^w(s_\star) = \sum_{f \in \mathcal{F}} w(f)[s_\star \models f] \leq 0. \tag{15.1}$$

It is consistent if and only if

$$h^w(s) \leq \mathit{cost}(o) + h^w(s[\![o]\!]) \tag{15.2}$$

holds for every state s and every operator o applicable in s. Both conditions are linear inequalities over feature weights, but the number of constraints (15.2) is exponential

in the task's size. We now show how to represent an equivalent condition in a more compact form.

We say an operator o affects a feature f in a state s if applying o in s changes the truth value of f, i.e. if $[s \models f] \neq [s[\![o]\!] \models f]$. Rewriting condition (15.2) shows that features unaffected by the applied operator cancel out.

$$\begin{aligned} \mathit{cost}(o) &\geq h^w(s) - h^w(s[\![o]\!]) \\ &= \sum_{f \in \mathcal{F}} w(f)[s \models f] - \sum_{f \in \mathcal{F}} w(f)[s[\![o]\!] \models f] \\ &= \sum_{f \in \mathcal{F}} w(f)([s \models f] - [s[\![o]\!] \models f]) \end{aligned}$$

If all features are atomic, the only features affected by o are atoms that are produced or consumed by o. In TNF an operator o produces the atoms $\mathit{eff}(o) \setminus \mathit{pre}(o)$ and consumes the atoms $\mathit{pre}(o) \setminus \mathit{eff}(o)$. All other features can be removed because their coefficients cancel out. However, we keep all atoms in $\mathit{pre}(o) \cup \mathit{eff}(o)$ to simplify the formula.

$$\begin{aligned} \mathit{cost}(o) &\geq \sum_{\substack{f \in \mathcal{F} \\ f \in \mathit{pre}(o) \cup \mathit{eff}(o)}} w(f)([s \models f] - [s[\![o]\!] \models f]) \\ &= \sum_{f \in \mathit{pre}(o) \cap \mathcal{F}} w(f) \quad - \sum_{f \in \mathit{eff}(o) \cap \mathcal{F}} w(f) \end{aligned}$$

This constraint no longer depends on the state s and still characterizes consistent potential heuristics with atomic features.

Theorem 15.1. *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, \mathit{cost} \rangle$ be a planning task in TNF and $\mathcal{F}$ a set of atomic features for Π. Let $C_\mathcal{F}$ be the following set of linear constraints over the variables $\{\mathsf{W}_f \mid f \in \mathcal{F}\}$.*

$$\sum_{f \in \mathcal{F}} \mathsf{W}_f[s_\star \models f] \leq 0$$

$$\sum_{f \in \mathit{pre}(o) \cap \mathcal{F}} \mathsf{W}_f \quad - \sum_{f \in \mathit{eff}(o) \cap \mathcal{F}} \mathsf{W}_f \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O}$$

The set of solutions for $C_\mathcal{F}$ is the set of weight functions of admissible and consistent potential heuristics for Π over $\mathcal{F}$.

If $\mathcal{A}$ is the set of atoms of Π, then the characterization requires at most $|\mathcal{A}|$ variables and only $|\mathcal{O}| + 1$ constraints. Both numbers are linear in the description size of Π. Since consistent heuristics are admissible iff they are goal-aware, we directly have the following corollary.

Corollary 15.1. *The set of admissible and consistent atomic potential heuristics can be characterized by a compact set of linear constraints.*

If $\mathcal{F}$ is the set of all atomic features, the weights can be normalized to simplify an implementation. Given a weight function w that satisfies the constraints, we define w' as

$$w'(\langle V, v\rangle) = w(\langle V, v\rangle) - w(\langle V, s_\star[V]\rangle)$$

This fixes the weight of all features that are true in $s_\star$ to 0. The goal state thus has a value of exactly 0 (satisfying the goal-awareness constraint). The heuristic values of $h^{w'}$ are the same as those of h^w after subtracting a constant value of $\sum_{V\in\mathcal{V}} w(\langle V, s_\star[V]\rangle)$. If a heuristic is consistent, then it remains consistent after subtracting a constant from every heuristic value. The normalized weight function w' thus also satisfies the consistency constraints. We can restrict the set of solutions to normalized weight functions by replacing the goal-awareness constraint with $\mathsf{W}_{\langle V, s_\star[V]\rangle} = 0$ for all variables V or by just removing all features true in $s_\star$ from $\mathcal{F}$.

15.2. Binary Potential Heuristics

Two-dimensional potential heuristics only consider atomic and binary features. As we discussed in the previous section, a potential heuristic is goal-aware if it satisfies

$$h^w(s_\star) = \sum_{f\in\mathcal{F}} w(f)[s_\star \models f] \leq 0,$$

and consistent if it satisfies

$$\mathit{cost}(o) \geq h^w(s) - h^w(s[\![o]\!]) = \sum_{f\in\mathcal{F}} w(f)([s \models f] - [s[\![o]\!] \models f])$$

for all states s and all operators o applicable in s. The goal-awareness constraint is a single linear constraint no matter what the set of features is. The consistency constraint, however, consists of an exponential number of inequalities (one for each transition). In the following we consider the inequalities for a specific operator o and abbreviate the change of a feature's truth value ($[s \models f] - [s[\![o]\!] \models f]$) as $\Delta_o(f, s)$.

In the atomic case, the coefficient $\Delta_o(f, s)$ is independent of the context in which o is applied and only dependent on o itself. For binary features, this is not the case. For example, consider the feature $f = \langle \textit{pos-T}, A\rangle \wedge \langle \textit{pos-P}, A\rangle$ in our running example task from Section 2.1. The feature expresses that the truck and the package are in position A. The operator *drive-B-A* that moves the truck from B to A produces f if it is applied in a state s_1 where the package already is at A and we have $\Delta_o(f, s_1) = -1$. However, if the same operator is applied in a state s_2 where the package is at location B, then f is false before and after applying the operator and $\Delta_o(f, s_2) = 0$.

We partition the set of features into three subsets for the operator o: *irrelevant* features $\mathcal{F}_o^{\text{irr}}$ have no variables in common with *vars*(o), *context-independent* features $\mathcal{F}_o^{\text{ind}}$ mention only variables in *vars*(o), and the remaining *context-dependent* features $\mathcal{F}_o^{\text{ctx}}$ mention one variable from *vars*(o) and one from $\mathcal{V} \setminus$ *vars*(o).

The truth value of an irrelevant feature never changes by applying o in some state. Thus, $\Delta_o(f_{\text{irr}}, s) = 0$ for all $f_{\text{irr}} \in \mathcal{F}_o^{\text{irr}}$ and states s in which o is applicable, and

$$\sum_{f \in \mathcal{F}_o^{\text{irr}}} w(f)\Delta_o(f, s) = 0. \tag{15.3}$$

For a context-independent feature f_{ind}, the effect of applying o in s is completely determined by o: f_{ind} holds in s iff f_{ind} is entailed by the precondition, and in $s[\![o]\!]$ iff it is entailed by the effect. Thus, $\Delta_o(f_{\text{ind}}, s) = [pre(o) \models f_{\text{ind}}] - [\mathit{eff}(o) \models f_{\text{ind}}]$ for every state s in which o is applicable. To emphasize that $\Delta_o(f_{\text{ind}}, s)$ does not depend on the state s for $f_{\text{ind}} \in \mathcal{F}_o^{\text{ind}}$, we abbreviate the notation to $\Delta_o(f_{\text{ind}})$ and get

$$\sum_{f \in \mathcal{F}_o^{\text{ind}}} w(f)\Delta_o(f, s) = \sum_{f \in \mathcal{F}_o^{\text{ind}}} w(f)\Delta_o(f). \tag{15.4}$$

A context-dependent feature $f_{\text{ctx}} \in \mathcal{F}_o^{\text{ctx}}$ is a conjunction $f_{\text{ctx}} = a_o \wedge a_{\bar{o}}$ where a_o is an atom over a variable in *vars*(o) and $a_{\bar{o}}$ is an atom over a variable in $\mathcal{V}_{\bar{o}} = \mathcal{V} \setminus$ *vars*(o). Applying o cannot change the truth value of $a_{\bar{o}}$. If o is applied in a state s with $s \not\models a_{\bar{o}}$, then $s \not\models f_{\text{ctx}}$ and $s[\![o]\!] \not\models f_{\text{ctx}}$, so $\Delta_o(f_{\text{ctx}}, s) = 0$. For the remaining situations, we know that $a_{\bar{o}}$ is present in both s and $s[\![o]\!]$ and the truth value of a_o is solely determined by o. Thus $\Delta_o(f_{\text{ctx}}, s) = \Delta_o(a_o)[s \models a_{\bar{o}}]$. If o is applicable in s,

$$\begin{aligned}
\sum_{f \in \mathcal{F}_o^{\text{ctx}}} w(f)\Delta_o(f, s) &= \sum_{\substack{f \in \mathcal{F}_o^{\text{ctx}} \\ f = a_o \wedge a_{\bar{o}}}} \sum_{\substack{V \in \mathcal{V}_{\bar{o}} \\ \mathit{vars}(a_{\bar{o}}) = \{V\}}} w(f)\Delta_o(f, s) && (15.5)\\
&= \sum_{\substack{f \in \mathcal{F}_o^{\text{ctx}} \\ f = a_o \wedge a_{\bar{o}}}} \sum_{V \in \mathcal{V}_{\bar{o}}} w(f)\Delta_o(a_o)[a_{\bar{o}} = \langle V, s[V]\rangle] && (15.6)\\
&= \sum_{V \in \mathcal{V}_{\bar{o}}} \sum_{\substack{f \in \mathcal{F}_o^{\text{ctx}} \\ f = a_o \wedge a_{\bar{o}}}} w(f)\Delta_o(a_o)[a_{\bar{o}} = \langle V, s[V]\rangle] && (15.7)\\
&\leq \sum_{V \in \mathcal{V}_{\bar{o}}} \sum_{\substack{f \in \mathcal{F}_o^{\text{ctx}} \\ f = a_o \wedge a_{\bar{o}}}} w(f)\Delta_o(a_o)[a_{\bar{o}} = \langle V, v_V^{\max}]\rangle] && (15.8)
\end{aligned}$$

where $v_V^{\max}$ is a value in *dom*(V) for which the inner sum is maximal, i.e.

$$v_V^{\max} \in \operatorname*{arg\,max}_{v \in \mathit{dom}(V)} \sum_{\substack{f \in \mathcal{F}_o^{\text{ctx}} \\ f = a_o \wedge a_{\bar{o}}}} w(f)\Delta_o(a_o)[a_{\bar{o}} = \langle V, v\rangle].$$

Let b^o_V be the inner sum in (15.8). Combining what we know about irrelevant features (15.3), context-independent features (15.4), and context-dependent features (15.8), we get

$$h^w(s) - h^w(s') = \sum_{f \in \mathcal{F}} w(f)\Delta_o(f,s) \leq \sum_{f \in \mathcal{F}^{\text{ind}}_o} w(f)\Delta_o(f) + \sum_{V \in \mathcal{V}_{\bar{o}}} b^o_V \tag{15.9}$$

for all states s and all operators o applicable in s. Therefore, if the weight function satisfies $\sum_{f \in \mathcal{F}^{\text{ind}}_o} w(f)\Delta_o(f) + \sum_{V \in \mathcal{V}_{\bar{o}}} b^o_V \leq \mathit{cost}(o)$ for all operators o, then h^w is consistent. Conversely, if h^w is consistent, then $h^w(s) - h^w(s[\![o]\!]) \leq \mathit{cost}(o)$ for operator o and the states s in which o is applicable. In particular, it is consistent for state $s^{\max}$ with $s^{\max}[V] = \mathit{pre}(o)[V]$ for $V \in \mathit{vars}(o)$, and $s^{\max}[V] = v^{\max}_V$ otherwise. It is then not difficult to check that the inequality in (15.9) is tight for such states $s^{\max}$. Hence, h^w is consistent iff

$$\sum_{f \in \mathcal{F}^{\text{ind}}_o} w(f)\Delta_o(f) + \sum_{V \in \mathcal{V}_{\bar{o}}} b^o_V \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O}$$

$$b^o_V = \max_{v \in \mathit{dom}(V)} \sum_{\substack{f \in \mathcal{F}^{\text{ctx}}_o \\ f = a_o \wedge \langle V, v\rangle}} w(f)\Delta_o(a_o) \qquad \text{for all } o \in \mathcal{O} \text{ and } V \in \mathcal{V}_{\bar{o}}.$$

Since b^o_V only occurs on the left-hand side of a single $\leq$-constraint, we can replace it with a new variable z^o_V and the constraints

$$z^o_V \geq \sum_{\substack{f \in \mathcal{F}^{\text{ctx}}_o \\ f = a_o \wedge \langle V, v\rangle}} w(f)\Delta_o(a_o) \qquad \text{for all } v \in \mathit{dom}(V).$$

it is easy to see that a valid value for b^o_V is also valid for z^o_V. A valid solution for z^o_V cannot be smaller than the maximum defining b^o_V. If it is larger, reducing it to be equal cannot make a previously satisfied constraint unsatisfied. The models including b^o_V and z^o_V thus have the same set of solutions projected to w. The resulting constraints are all linear, and we can put the different parts together:

Theorem 15.2. *Let $\mathcal{F}$ be a set of features of size at most 2 for a TNF planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, \mathit{cost}\rangle$. Let $C_{\mathcal{F}}$ be the following set of linear constraints over the variables $\{\mathsf{W}_f \mid f \in \mathcal{F}\} \cup \{\mathsf{Z}^o_V \mid o \in \mathcal{O}, V \in \mathcal{V}_{\bar{o}}\}$.*

$$\sum_{f \in \mathcal{F}} \mathsf{W}_f[s_\star \models f] \leq 0$$

$$\sum_{f \in \mathcal{F}^{\text{ind}}_o} \mathsf{W}_f\Delta_o(f) + \sum_{V \in \mathcal{V}_{\bar{o}}} \mathsf{Z}^o_V \leq \mathit{cost}(o) \qquad \textit{for all } o \in \mathcal{O}$$

$$\sum_{\substack{f \in \mathcal{F}^{\text{ctx}}_o \\ f = a_o \wedge \langle V, v\rangle}} \mathsf{W}_f\Delta_o(a_o) \leq \mathsf{Z}^o_V \qquad \textit{for all } V \in \mathcal{V}_{\bar{o}},\ v \in \mathit{dom}(V), \textit{ and } o \in \mathcal{O}$$

where $\Delta_o(f) = [pre(o) \models f] - [\mathit{eff}(o) \models f]$. *The set of solutions for* $C_\mathcal{F}$ *projected to* $\mathbf{W}$ *is the set of weight functions of admissible and consistent potential heuristics for* Π *over* $\mathcal{F}$.

The model $C_\mathcal{F}$ has $O(|\mathcal{F}| + |\mathcal{O}||\mathcal{V}|)$ variables and $O(|\mathcal{O}||\mathcal{V}|d)$ constraints where d bounds the size of the variable domains, and each constraint has $O(|\mathcal{F}| + |\mathcal{V}|)$ coefficients.

Corollary 15.2. *The set of admissible and consistent binary potential heuristics can be characterized by a compact set of linear constraints.*

Analogously to the atomic case, weight functions can be normalized so the weight of every feature that is true in the goal state is 0. For potential heuristics over the set of all binary features, this only excludes solutions that match a normalized solution shifted by an additive constant.

15.3. Higher-Dimensional Potential Heuristics

We now prove that a general result like Corollary 15.2 is not possible for sets of features of dimension 3 or more, unless NP equals P. However, we identify classes of problems on which linear constraints can compactly characterize higher-dimensional potential heuristics.

15.3.1. Intractability

Corollary 15.2 allows one to answer many interesting questions about binary potential heuristics in polynomial time. In particular, by solving a single LP one can test whether a given potential function is consistent and/or goal-aware. We use this idea to show that no general result like Corollary 15.2 is possible for potential heuristics of dimension 3 or more. The proof is a reduction of non-3-colorability, a decision problem that is complete for co-NP (Garey and Johnson, 1979), to the problem of testing whether a potential function of dimension 3 is consistent.

Let $G = \langle V, E\rangle$ be an undirected graph. We first construct, in polynomial time, a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathrm{I}, s_\star, \mathit{cost}\rangle$ in TNF and a potential function φ of dimension 3 such that G is not 3-colorable iff φ is consistent. The task Π has $|V| + 1$ variables: one variable C_v for the color of each vertex $v \in V$ that can be either red, blue or green, and one "master" binary variable denoted by M.

For every vertex $v \in V$ and pair of different colors $c, c' \in \mathit{dom}(C_v)$, there is a unique operator $o_{v,c,c'}$ of zero cost that changes C_v from c to c' when $M = 0$. For the variable M, there is a unique operator o_M, also of zero cost, that changes M from 0 to 1. These are all the operators in the task Π.

Each state $s \in \mathcal{S}$ encodes a coloring of G, where the color of vertex v is the value $s[C_v]$ of the state variable C_v. The initial state s_{I} is set to an arbitrary coloring but with the master variable set to 0; e.g. $s_{\mathrm{I}}[M] = 0$ and $s_{\mathrm{I}}[C_v] = red$ for every vertex $v \in V$. The goal state $s_\star$ is also set to an arbitrary coloring but with $s_\star[M] = 1$; e.g. $s_\star[M] = 1$ and $s_\star[C_v] = red$ for every vertex $v \in V$.

The potential function φ of dimension 3 is constructed as follows. For features f with $\mathit{vars}(f) = \{M, C_u, C_v\}$ such that $\{u, v\} \in E$ is an edge in the graph, let its weight $w(f) = -1$ when $f[M] = 1$ and $f[C_u] \neq f[C_v]$, and $w(f) = 0$ otherwise. For the feature $f_M = \langle M, 1\rangle$ of dimension 1, let $w(f_M) = |E| - 1$. The weight $w(f)$ for all other features f is set to 0.

Let us now reason about the states of the task Π and the values assigned to them by the potential function φ. Let s be a state for Π. If $s[M] = 0$, then $\varphi(s) = 0$. If $s[M] = 1$, then no operator is applicable at s, and $\varphi(s) \geq -1$. For such states, the feature f_M contributes a value of $|E| - 1$ to $\varphi(s)$, while the features corresponding to edges contribute a value of $-|E|$ when s encodes a coloring. Therefore, $\varphi(s) = -1$ iff s encodes a 3-coloring of G .

Let us consider a transition $s \xrightarrow{o} s' \in \mathcal{T}$. Clearly, $s[M] = 0$ as no operator is applicable in states with $s[M] = 1$. If $s[M] = s'[M] = 0$, then $\varphi(s) = \varphi(s') = 0$. All operator costs are equal to zero, so φ is always consistent for transitions like this. If $s[M] = 0$ and $s'[M] = 1$, then $\varphi(s) \leq \varphi(s')$ iff s' does not encode a 3-coloring of G. Therefore, φ is consistent iff there is no transition $s \xrightarrow{o} s'$ with $s[M] = 0$, $s'[M] = 1$ and s' encoding a 3-coloring of G. Thus, φ is consistent iff the graph G is not 3-colorable.

Finally, observe that testing whether φ is inconsistent is possible in non-deterministic polynomial time: guess a state s and an operator o, and check whether $\varphi(s) > \varphi(s[\![o]\!])$.

Theorem 15.3. *Let $\mathcal{F}$ be a set of features for a planning task Π, and let φ be a potential function over $\mathcal{F}$. Testing whether φ is consistent is co-NP-complete.*

Looking at the derivation of constraints for the binary case in Section 15.2, we notice that only part of the proof relies on the features being binary. Intutively, the constraints for a context-dependent feature split the effect of an operator into the part completely determined by the operator and the part that depends on the state. The state-dependent part is then replaced by the maximum over all possible states. In the binary case, the maximum can be represented by a compact set of linear constraints. Theorem 15.3 implies that this is not possible in general. But there is a silver lining: we now prove a parametrized tractability result for higher-dimensional potentials heuristics.

15.3.2. Parametrized Tractability

Of the two conditions that characterize goal-aware and consistent potential heuristics, goal-awareness is always easy to specify as a linear constraint.

$$h^w(s_\star) = \sum_{f \in \mathcal{F}} w(f)[s_\star \models f] \leq 0,$$

Consistency is more challenging to test as it consists of an exponential number of inequalities, one per state. The constraint is equivalent to

$$\mathit{cost}(o) \geq \max_{s \models \mathit{pre}(o)} (h^w(s) - h^w(s[\![o]\!])) = \max_{s \models \mathit{pre}(o)} \sum_{f \in \mathcal{F}} w(f) \Delta_o(f, s) \tag{15.10}$$

for all operators $o \in \mathcal{O}$.

As done before, we partition $\mathcal{F}$ as $\mathcal{F} = \mathcal{F}_o^{\text{irr}} \cup \mathcal{F}_o^{\text{ind}} \cup \mathcal{F}_o^{\text{ctx}}$ for a fixed operator o, where $\mathcal{F}_o^{\text{irr}}$ contains the features that have no variable in common with $\mathit{vars}(o)$, $\mathcal{F}_o^{\text{ind}}$ contains the features that use only variables from $\mathit{vars}(o)$, and $\mathcal{F}_o^{\text{ctx}}$ contains the remaining features.

As for binary features, $\Delta_o(f_{\text{irr}}, s) = 0$ for all $f_{\text{irr}} \in \mathcal{F}_o^{\text{irr}}$ and states s in which o is applicable. The value of Δ_o also does not depend on s for features in $\mathcal{F}_o^{\text{ind}}$ and we write $\Delta_o(f_{\text{ind}}, s)$ as $\Delta_o(f_{\text{ind}}) = [\mathit{pre}(o) \models f_{\text{ind}}] - [\mathit{eff}(o) \models f_{\text{ind}}]$ for all $f_{\text{ind}} \in \mathcal{F}_o^{\text{ind}}$ and states s in which o is applicable. We can thus rewrite constraint (15.10) by taking state-independent parts out of the maximum:

$$\mathit{cost}(o) \geq \sum_{f \in \mathcal{F}_o^{\text{ind}}} w(f) \Delta_o(f) + \max_{s \models \mathit{pre}(o)} \sum_{f \in \mathcal{F}_o^{\text{ctx}}} w(f) \Delta_o(f, s) \tag{15.11}$$

In the remaining state-dependent part, state s is partly determined by $\mathit{pre}(o)$. For a given feature f, the value of $\Delta_o(f, s)$ therefore only depends on variables in $\mathcal{V}_{f \setminus o} = \mathit{vars}(f) \setminus \mathit{vars}(o)$. The feature is true in s iff $s|_{\mathit{vars}(f)} \models f$ iff $(\mathit{pre}(o) \cup s|_{\mathcal{V}_{f \setminus o}}) \models f$. Analogously, $s[\![o]\!] \models f$ iff $(\mathit{eff}(o) \cup s|_{\mathcal{V}_{f \setminus o}}) \models f$. The coefficient of $w(f)$ thus only depends on the values of the variables $\mathcal{V}_{f \setminus o}$. Unfortunately, these sets can overlap for different features in the sum. We call this a *function maximization problem*: given a set of scoped functions Ψ, where the scope of each function is a subset of a common set of finite domain variables, we are looking for an assignment to these variables such that the sum of all functions is maximal. The solution of this problem is denoted $\mathit{Max}(\Psi)$ with

$$\mathit{Max}(\Psi) = \max_{\nu \in \mathit{dom}(\mathcal{V})} \sum_{\langle S, \psi \rangle \in \Psi} \psi(\nu|_S).$$

Computing $\mathit{Max}(\Psi)$ is the goal of constraint optimization for extensional constraints, an important problem in AI. It is challenging because the number of valuations over the variables is exponential in the number of variables. Usually, the functions in Ψ map to numbers but in Appendix C we show that a generalized version of the bucket elimination algorithm (Dechter, 2003) can handle functions that map to linear expressions.

We use LP variables $\{\mathsf{W}_f \mid f \in \mathcal{F}\}$ to represent the weights and define the following function for each feature f that maps partial variable assignments over $\mathcal{V}_{f \setminus o}$ to $\{\mathsf{W}_f, -\mathsf{W}_f, 0\}$:

$$\psi_o^f(p) = \mathsf{W}_f([\mathit{pre}(o) \cup p \models f] - [\mathit{eff}(o) \cup p \models f])$$

The state-dependent part of constraint 15.11 is then equivalent to the function maximization problem $\Psi_o = \{\langle \mathcal{V}_{f \setminus o}, \psi_o^f \rangle \mid f \in \mathcal{F}_o^{\text{ctx}}\}$. Constraint 15.11 can be written as the combination of $\mathit{Max}(\Psi_o)$ and linear constraints:

$$\mathit{cost}(o) \geq \sum_{f \in \mathcal{F}_o^{\text{ind}}} \mathsf{W}_f \Delta_o(f) + \mathsf{Z}_o \tag{15.12}$$

$$\mathsf{Z}_o = \mathit{Max}(\Psi_o) \tag{15.13}$$

As we prove in Appendix C (Corollary C.1 and Theorems C.1 and C.2), the bucket elimination algorithm can be used to generate a set of linear constraints $P_{\text{LP}(o)}$ which is equivalent to $\mathsf{Z}_o \geq \mathit{Max}(\Psi_o)$. Since Z_o only occurs in constraint (15.13) and with a positive coefficient on the right-hand side of constraint (15.12), the inequality is sufficient for our purpose.

Theorem 15.4. *Let $\mathcal{F}$ be a set of features for a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\text{I}}, s_\star, \mathit{cost} \rangle$ in TNF. For each operator $o \in \mathcal{O}$, let $P_{\text{LP}(o)}$ be the set of constraints generated by the bucket elimination algorithm which is equivalent to $\mathsf{Z}_o \geq \mathit{Max}(\Psi_o)$.*

Let $C_{\mathcal{F}}$ be the following set of linear constraints over the auxiliary variables in $P_{\text{LP}(o)}$ for all operators o and variables $\{\mathsf{W}_f \mid f \in \mathcal{F}\}$.

$$\sum_{f \in \mathcal{F}} \mathsf{W}_f[s_\star \models f] \leq 0$$

$$\sum_{f \in \mathcal{F}_o^{\text{ind}}} \mathsf{W}_f \Delta_o(f) + \mathsf{Z}_o \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O}$$

$$P_{\text{LP}(o)} \qquad \text{for all } o \in \mathcal{O}$$

where $\Delta_o(f) = [\mathit{pre}(o) \models f] - [\mathit{eff}(o) \models f]$. The set of solutions for $C_{\mathcal{F}}$ projected to $\mathbf{W}$ is the set of weight functions of admissible and consistent potential heuristics for Π over $\mathcal{F}$.

We finish the section by bounding the number and size of the constraints in $C_{\mathcal{F}}$ which is mostly determined by the variables W_f for every feature and the combined size of all $P_{\text{LP}(o)}$. For an operator o, the size of $P_{\text{LP}(o)}$ depends on how interconnected the features in $\mathcal{F}_o^{\text{ctx}}$ are and on the order in which variables are eliminated in the bucket elimination algorithm for $\mathit{Max}(\Psi_o)$. To quantify this, we define the *context-dependency graph* $G(\Pi, \mathcal{F}, o)$ for a task Π, features $\mathcal{F}$, and operator o as follows: the vertices are the variables of Π. Edges connect vertices V and V' when there is a feature $f \in \mathcal{F}$ such that $\{V, V'\} \subseteq \mathit{vars}(f) \setminus \mathit{vars}(o)$. For each variable elimination order σ, the size of the resulting constraints $P_{\text{LP}(o)}$ can be expressed in terms of a parameter called *induced width* $w(\sigma)$. The width induced by the best elimination order is the treewidth of $G(\Pi, \mathcal{F}, o)$ (Dechter, 2003). In Theorem C.3 of Appendix C we show that $P_{\text{LP}(o)}$ has $O(|\mathcal{V}| d^{w(\sigma)})$ auxiliary variables and $O(|\mathcal{V}| d^{w(\sigma)+1})$ constraints, where d bounds the size of the variable domains and $w(\sigma)$ is the induced width of $G(\Psi_o)$ and a variable order σ.

Proposition 15.1. *Let $\mathcal{F}$ be a set of features for a TNF planning task Π with operators $\mathcal{O}$ and variables $\mathcal{V}$. Let $\{\sigma_o\}_{o\in\mathcal{O}}$ be an indexed collection of orderings on $\mathcal{V}$.*

The set of admissible and consistent potential heuristics over features $\mathcal{F}$ can be characterized by a set of $O(|\mathcal{O}||\mathcal{V}|d^{w+1})$ linear constraints over $O(|\mathcal{F}| + |\mathcal{O}||\mathcal{V}|d^w)$ variables, where d bounds the size of the variable domains, and w is the maximal induced width of $G(\Pi, \mathcal{F}, o)$ along σ_o for any $o \in \mathcal{O}$.

Since the optimal induced width of a graph is its treewidth, we can conclude the following fixed-parameter tractability result.

Corollary 15.3. *Constructing linear constraints that characterize the set of admissible and consistent potential heuristics for a set of features $\mathcal{F}$ is fixed-parameter tractable with the parameter $\max(d, w)$, where d bounds the size of the variable domains, and w is the the maximal treewidth of the context-dependency graph for an operator.*

If we restrict the set of features in a way that the treewidth of context-dependency graphs is limited by a constant, there always is a polynomial-sized set of linear constraints characterizing admissible and consistent potential heuristics. However, finding this set depends on using a good variable order in the bucket elimination algorithms, and finding a variable order that minimizes the induced width of a graph is an NP-hard problem in itself (Dechter, 2003).

Let us apply Proposition 15.1 when $\mathcal{F}$ is a set of binary features. In this case, $G(\Pi, \mathcal{F}, o)$ has no edges for all $o \in \mathcal{O}$ and thus its induced width along any order is 0. Proposition 15.1 asserts that the number of constraints is $O(|\mathcal{O}||\mathcal{V}|d^{w+1}) = O(|\mathcal{O}||\mathcal{V}|d)$, while their size is $O(|\mathcal{F}| + |\mathcal{V}|d^w) = O(|\mathcal{F}| + |\mathcal{V}|)$ agreeing with Corollary 15.2.

15.4. Objective Functions

For certain sets of features $\mathcal{F}$, we can characterize admissible and consistent potential functions over $\mathcal{F}$ with a compact set of linear constraints $C_\mathcal{F}$. Given any objective function *obj*, the weights that maximize *obj* subject to $C_\mathcal{F}$ describe a "best" heuristic function according to *obj*. If *obj* is a linear combination of feature weights, we can use an LP solver to come up with such weights. But how do we define "best"?

We now explore different ways to measure heuristic quality. Most of them rely on the observation that the heuristic value of a state s is a linear combination of feature weights:

$$h(s) = \sum_{f\in\mathcal{F}} w(f)[s \models f]$$

For admissible heuristics, higher heuristic values are generally better. We can easily find a potential heuristic with the highest possible value for the initial state s_I by

maximizing $h(s_\mathrm{I})$ subject to $C_\mathcal{F}$. The resulting heuristic, which we call $h^{\mathrm{pot}}_{\mathcal{F},s_\mathrm{I}}$ has some interesting theoretical properties, which we discuss in Chapter 16.

One obvious disadvantage of maximizing the heuristic value of only one state is that there is no incentive to optimize potentials of features that do not occur in the state. We therefore introduce alternative objective functions that consider more than one state.

Instead of maximizing the heuristic value of a single state, we can maximize the average heuristic value of multiple states. In general, the average heuristic value for any set of states S is a weighted sum over potentials:

$$\frac{1}{|S|}\sum_{s\in S} h(s) = \frac{1}{|S|}\sum_{s\in S}\sum_{f\in\mathcal{F}} w(f)[s \models f]$$

Note that we can generally eliminate any linear transformation of the objective function since we are not interested in the optimal objective value itself. It makes no difference if we optimize the average heuristic value, or the sum of heuristic values, in S. For example, if S is a reasonably small set of sampled states, the coefficient of $w(f)$ can be set to number of states in which f is true to maintain the fact that all coefficients are integers. If we consider the set of all states $\mathcal{S}$, the number of states in which a feature is true can become very large. To avoid numeric problems, we can use the proportion of states in which the feature is true, instead. If $\mathit{dom}(\mathit{vars}(f))$ is the product of $\mathit{dom}(V)$ for all $V \in \mathit{vars}(f)$, then for every partial variable assignment of variables $\mathcal{V} \setminus \mathit{vars}(f)$, there is one state in which f is true and $|\mathit{dom}(\mathit{vars}(f))| - 1$ states in which it is not. The proportion of states where f is true is thus $\frac{1}{\mathit{dom}(\mathit{vars}(f))}$ and we get:

$$\begin{aligned}
\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} h(s) &= \frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}} w(f)[s \models f] \\
&= \sum_{f\in\mathcal{F}} \frac{\sum_{s\in\mathcal{S}}[s \models f]}{|\mathcal{S}|} w(f) \\
&= \sum_{f\in\mathcal{F}} \frac{|\{s \in \mathcal{S} \mid s \models f\}|}{|\mathcal{S}|} w(f) \\
&= \sum_{f\in\mathcal{F}} \frac{1}{|\mathit{dom}(\mathit{vars}(f))|} w(f)
\end{aligned}$$

Maximizing this function subject to $C_\mathcal{F}$ yields an admissible potential heuristic over $\mathcal{F}$ with the highest possible average heuristic value, but there are two problems in practice.

First, if $\mathcal{S}$ contains dead ends, heuristic values can become arbitrarily large and the linear program can become unbounded. This can even happen if all dead ends are unreachable. When the linear program is unbounded, we usually cannot extract a useful heuristic function. Unfortunately, the LP is not always unbounded if $\mathcal{S}$ contains a dead

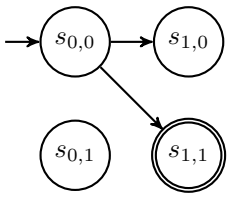


Figure 15.1: Example for a task where the set of all states $\mathcal{S}$ contains dead ends, but the LP maximizing the average heuristic value over all atomic features is bounded. There are two variables X and Y, each with the domain $\{0, 1\}$. A state $s_{x,y}$ assigns x to X and y to Y. Independently of the dead ends' reachability, the average heuristic value is bounded by finite heuristic values.

end so we cannot use this to test for dead ends in a set of states. Figure 15.1 shows an example of a task where the LP is bounded even though there are dead ends. In this task the sum of heuristic values is two times the weight of each atomic feature since every atom occurs in exactly two states. On the other hand, the heuristic values of $s_{0,0}$ and $s_{1,1}$ also add up to the sum over all feature weights, and those states must have a finite value if the heuristic is admissible. Thus, the optimal objective value must be bounded by

$$\sum_{s \in \mathcal{S}} h^w(s) = 2 \sum_{f \in \mathcal{F}} w(f) = 2(h^w(s_{0,0}) + h^w(s_{1,1})) < \infty.$$

Second, the heuristic values of unreachable states influence the solution. This is problematic since unreachable states are never encountered during the search. Thus it is pointless to optimize their heuristic value. Moreover, they can be fundamentally different from reachable states. For example, an invariant analysis can detect atoms that can never occur together in a reachable state, i.e. they are mutex. The set of all states $\mathcal{S}$ also includes states that violate such mutex information, and maximizing its average heuristic value requires considering them. We would like to only consider reachable states, but we cannot characterize this set of states concisely. (If this could be done efficiently, it would also present an efficient test for plan existence as we could use it to check if the goal is reachable.) While we could exclude all states that violate a single mutex, excluding multiple (potentially overlapping) mutexes is more complicated.

The negative influence of dead ends can be handled to some extent by introducing an upper bound M on each potential. If $w(f) \leq M$ for all $f \in \mathcal{F}$, then the heuristic value of each state is also limited by $|\mathcal{F}|M$. When optimizing for an individual state s, unlimited weights may identify s as a dead end (if the LP is unbounded). While this is no longer the case when weights are limited, it has the benefit that we can maximize the average heuristic value of any set of states and are always able to extract a heuristic function. If M is large enough, "recognized" dead ends have heuristic values higher than $h^*(s_{\mathrm{I}})$ and are never expanded.

Ignoring unreachable states is a tougher problem. Ideally, we would like to maximize the heuristic values of reachable state and completely ignore all unreachable states, but even detecting whether a state is unreachable is as hard as planning itself. As an approximation, it is possible to randomly sample states from the reachable part of the state space and maximize the average heuristic value of these samples. If the sampled states accurately represent the reachable state space, a potential function that maximizes their average heuristic value can generalize to other states.

Seipp, Pommerening, and Helmert (2015) also experiment with functions that approximate search effort. Assume there is a function *effort*(h) that correctly predicts the number of nodes that have to be expanded with an A* search using heuristic h. Minimizing *effort*(h) subject to $C_{\mathcal{F}}$ would yield the admissible potential heuristic over $\mathcal{F}$ that is best suited to the particular search space. The problem, of course, is that *effort*(h) is not known in practice and is hard to approximate accurately. There are approximations for IDA* search (Korf, Reid, and Edelkamp, 2001) but they are based on assumptions that do not necessarily hold for A* search. A reasonably good approximation of *effort*(h) is also not likely to be a linear function of the feature weights of h. Seipp, Pommerening, and Helmert (2015) consider an approximation that is quadratic in the weights and find weights for atomic potential heuristics with quadratic programming. We will later see that simpler methods already achieve a heuristic quality very close to the theoretical limit for atomic potential heuristics. The additional time spent on solving a quadratic program thus does not pay off. However, on a theoretical level this remains an interesting question for future research.

16. Theoretical Analysis

We now connect potential heuristics to the two previous parts by analyzing their relationship to operator counting and to cost partitioning. It turns out that the three ideas are very closely related.

16.1. Connection to Operator Counting

We first look at a special case for atomic potential heuristics and their connection to operator counting. This connection can be generalized, but we will do so in the following section using the connection between operator counting and cost partitioning.

Consider the set of all atomic features $\mathcal{F} = \mathcal{A}$, i.e. all atoms. Optimal solutions of the following model are weight functions of consistent and admissible potential heuristics over $\mathcal{F}$ that maximize the potential of the initial state s_{I}.

$$\begin{aligned}
&\text{Maximize } \sum_{f \in \mathcal{F}} \mathsf{W}_f[s_{\mathrm{I}} \models f] \text{ subject to} \\
&\sum_{f \in \mathcal{F}} \mathsf{W}_f[s_\star \models f] \leq 0 \\
&\sum_{f \in \mathit{pre}(o) \cap \mathcal{F}} \mathsf{W}_f \; - \sum_{f \in \mathit{eff}(o) \cap \mathcal{F}} \mathsf{W}_f \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O}
\end{aligned}$$

Because $\mathcal{F}$ is exactly the set of atoms, we can write the model in terms of atoms $\langle V, v \rangle$:

$$\begin{aligned}
&\text{Maximize } \sum_{\langle V,v \rangle \in s_{\mathrm{I}}} \mathsf{W}_{\langle V,v \rangle} \text{ subject to} \\
&\sum_{\langle V,v \rangle \in s_\star} \mathsf{W}_{\langle V,v \rangle} \leq 0 \\
&\sum_{\langle V,v \rangle \in \mathit{pre}(o)} \mathsf{W}_{\langle V,v \rangle} \; - \sum_{\langle V,v \rangle \in \mathit{eff}(o)} \mathsf{W}_{\langle V,v \rangle} \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O}
\end{aligned}$$

As we remarked in Section 15.1, we can restrict our attention to normalized potential heuristic where $w(f) = 0$ if $s_\star \models f$ when $\mathcal{F} = \mathcal{A}$. If w is the weight function of a non-normalized potential heuristic h^w, then w' with $w'(\langle V, v \rangle) = w(\langle V, v \rangle) - w(\langle V, s_\star[V] \rangle)$ is a weight function of a normalized potential heuristic $h^{w'}(s) = h^w(s) - h^w(s_\star)$.

If h^w is consistent and admissible, then so is $h^{w'}$. Since goal-awareness means that $h^w(s_\star) \leq 0$, this also implies that $h^{w'}$ dominates h^w. If we maximize the potential of a single state, as we do here, we can thus restrict attention to normalized potential heuristics:

$$\text{Maximize} \sum_{\langle V,v\rangle \in s_\mathrm{I}} \mathsf{W}_{\langle V,v\rangle} \text{ subject to}$$
$$\mathsf{W}_{\langle V,v\rangle} = 0 \qquad \text{for all } \langle V,v\rangle \in s_\star$$
$$\sum_{\langle V,v\rangle \in pre(o)} \mathsf{W}_{\langle V,v\rangle} - \sum_{\langle V,v\rangle \in \mathit{eff}(o)} \mathsf{W}_{\langle V,v\rangle} \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O}$$

Consider the dual of the resulting LP where Count_o is the dual variable for the constraint of operator o and $\mathsf{G}_{\langle V,v\rangle}$ is the dual variable corresponding to the goal-awareness constraint for $\langle V,v\rangle$:

$$\text{Minimize} \sum_{o\in\mathcal{O}} \mathsf{Count}_o \mathit{cost}(o) \text{ subject to}$$
$$\sum_{\substack{o\in\mathcal{O}\\ \langle V,v\rangle \in pre(o)}} \mathsf{Count}_o - \sum_{\substack{o\in\mathcal{O}\\ \langle V,v\rangle \in \mathit{eff}(o)}} \mathsf{Count}_o + \mathsf{G}_{\langle V,v\rangle}[s_\star[V]=v] = [s_\mathrm{I}[V]=v] \quad \text{for all } \langle V,v\rangle \in \mathcal{A}$$
$$\mathsf{Count}_o \geq 0 \qquad \text{for all } o \in \mathcal{O}$$

The variables $\mathsf{G}_{\langle V,v\rangle}$ are unrestricted and each occurs only in a single constraint, which makes those constraints redundant. We can therefore remove the constraints for $\langle V,v\rangle \in s_\star$ from the LP without affecting its solutions projected to the operator-counting variables. Note that this is the same as removing features that are true in the goal from the potential heuristic instead of forcing their value to zero during normalization. The remaining constraints are exactly the net change constraints for all atoms that do not occur in the goal. We have already seen in Section 10.3 that net change constraints for all values of a variable except one, imply the missing one. We can thus write the LP in the equivalent form that includes the implied constraint for each variable:

$$\text{Minimize} \sum_{o\in\mathcal{O}} \mathsf{Count}_o \mathit{cost}(o) \text{ subject to}$$
$$\sum_{\substack{o\in\mathcal{O}\\ \langle V,v\rangle \in pre(o)}} \mathsf{Count}_o - \sum_{\substack{o\in\mathcal{O}\\ \langle V,v\rangle \in \mathit{eff}(o)}} \mathsf{Count}_o = [s_\mathrm{I}[V]=v] - [s_\star[V]=v] \quad \text{for all } \langle V,v\rangle \in \mathcal{A}$$
$$\mathsf{Count}_o \geq 0 \qquad \text{for all } o \in \mathcal{O}$$

We have also shown in Section 10.3 that upper bound net change constraints are redundant in the presence of all lower bound net change constraints for a variable. Changing $=$ to $\geq$ in the constraints above thus does not affect the set of solutions. The resulting LP is an operator-counting LP that minimizes the total cost of operators subject to lower bound net change constraints for all atoms. We already know it as the model of the state equation heuristic.

Theorem 16.1. *Let Π be a planning task with atoms $\mathcal{A}$ and s one of its states. Let h^w be a consistent and admissible potential heuristic over the features $\mathcal{A}$ with maximal $h^w(s)$. Then $h^w(s) = h^{\text{SEQ}}(s)$.*

There are several interesting consequences of this result. For one, it shows that there is a close relation between operator-counting and potential heuristics. In the next section, we will see that this connection is not restricted to atomic features. The theorem also implies an important dominance result. The potential heuristic h^w achieves the best possible value for the state s mentioned in the theorem. Every other atomic potential heuristic $h^{w'}$ has $h^{w'}(s) \leq h^w(s) = h^{\text{SEQ}}(s)$. The state equation heuristic thus achieves the best value that can be achieved by *any* admissible and consistent potential heuristic over features $\mathcal{A}$ in each state.

Corollary 16.1. *Let Π be a planning task with atoms $\mathcal{A}$ and h^w a consistent and admissible potential heuristic over the features $\mathcal{A}$. Then h^{SEQ} dominates h^w.*

The main difference between h^{SEQ} and a potential heuristic over $\mathcal{A}$ is that the potential heuristic is computed by maximizing an objective *once* while h^{SEQ} maximizes its objective *for every state*. We can interpret the state equation heuristic as optimizing a new potential heuristic in each state, but using the weights only to compute the heuristic value of this state.

Corollary 16.2. *Let Π be a planning task with atoms $\mathcal{A}$ and for each state $s \in \mathcal{S}$ let h^w_s be a consistent and admissible potential heuristic over the features $\mathcal{A}$ that maximizes the potential of s. Then h^{SEQ} is the maximum over h^w_s for all $s \in \mathcal{S}$.*

This corollary gives an interpretation of using multiple potential heuristics where each is optimized to maximize the potential of a sampled state. This method approximates $h^{\text{SEQ}} = \max_{s \in \mathcal{S}} h^w_s$ by replacing $\mathcal{S}$ in the maximization with a smaller set of sampled states $\hat{S}$. The more closely $\hat{S}$ represents $\mathcal{S}$, the closer the resulting heuristic is to the upper bound given by h^{SEQ}.

The relation of potential heuristics and h^{SEQ} already gives us some insight on the connection to cost partitioning, as we know that h^{SEQ} computes an optimal general cost partitioning over atomic flow heuristics. We will now see how this relation generalizes to other potential heuristics.

16.2. Connection to Cost Partitioning

We have seen that an admissible and consistent potential heuristic over all atomic features that achieves the maximal initial heuristic value computes an optimal operator cost partitioning over atomic flow heuristics. We now extend this result to all potential heuristics that use the abstract states of a set of abstractions as features and show that they correspond to an optimal *transition cost partitioning* over the flow heuristics. For

abstractions without dead states flow and abstraction heuristics are identical, so this result also shows the relationship to cost-partitioned abstractions.

In Chapter 4 we discussed the LP model by Katz and Domshlak (2010b) to compute the optimal operator cost partitioning over a collection of abstractions A. It is easy to adapt this model to compute an optimal transition cost partitioning instead: instead of variables C^α_o that represent the cost of operator o attributed to abstraction α, we use variables $\mathsf{C}^\alpha_{s \xrightarrow{o} s'}$ that represent the cost of transition $s \xrightarrow{o} s'$ attributed to α. We are interested in cost-partitioned flow heuristic, not abstraction heuristics, so we have to include constraints on dead states.

Definition 16.1 (TCP heuristics)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathrm{I}, s_\star cost \rangle$ be a planning task with transitions $\mathcal{T}$, and let A be a set of abstractions for Π. Then $h^{\mathrm{TCP}}_{f,A}$ is the heuristic mapping a state s to the optimal objective value of the following LP or to ∞ if the LP is unbounded.*

$$\textit{Maximize} \sum_{\alpha \in A} \mathsf{H}^\alpha_{\alpha(s)} \textit{ subject to}$$

$$\mathsf{H}^\alpha_{\alpha(s_\star)} = 0 \qquad \textit{for all } \alpha \in A \tag{16.1}$$

$$\mathsf{H}^\alpha_{\alpha(s')} \leq \mathsf{H}^\alpha_{\alpha(s'')} + \mathsf{C}^\alpha_{s' \xrightarrow{o} s''} \qquad \textit{for all } \alpha \in A \textit{ and } s' \xrightarrow{o} s'' \in \mathcal{T}^\alpha \tag{16.2}$$

$$\sum_{\alpha \in A} \mathsf{C}^\alpha_{s \xrightarrow{o} s'} \leq \mathit{cost}(o) \qquad \textit{for all } s \xrightarrow{o} s' \in \mathcal{T} \tag{16.3}$$

The heuristic $h^{\mathrm{TCP}}_{h,A}$ maps each state to the optimal objective value of an LP with the same constraints except that (16.2) *only occurs if $\alpha(s')$ and $\alpha(s'')$ are alive in the abstract transition system* TS^α.

Proposition 16.1. *For a set of abstractions A the heuristic $h^{\mathrm{TCP}}_{h,A}$ is the optimal general transition cost partitioning heuristic for the abstraction heuristics h^α for $\alpha \in A$, and $h^{\mathrm{TCP}}_{f,A}$ is the optimal general transition cost partitioning heuristic for the flow heuristics f^α for $\alpha \in A$.*

Proof sketch: In both cases, constraint (16.3) ensures that the transition cost functions cost^α encoded in $\mathbf{C}^\alpha$ respect the cost partitioning property. The cost-partitioned heuristics map to the optimal objective value of the following LP for every $\alpha \in A$, to ∞ if it is unbounded, and to $-\infty$ if it is infeasible:

$$\text{Maximize } \mathsf{H}^\alpha_{\alpha(s)} \text{ subject to}$$

$$\mathsf{H}^\alpha_{\alpha(s_\star)} = 0 \tag{16.4}$$

$$\mathsf{H}^\alpha_{\alpha(s')} \leq \mathsf{H}^\alpha_{\alpha(s'')} + \mathit{cost}^\alpha(s' \xrightarrow{o} s'') \qquad \begin{array}{r}\text{for all } s' \xrightarrow{o} s'' \in \mathcal{T}^\alpha \\ (\text{if } \alpha(s') \text{ and } \alpha(s'') \text{ are alive})\end{array} \tag{16.5}$$

The proof for abstraction heuristics then is analogous to the one for the optimal operator cost partitioning. Constraints (16.4) and (16.5) ensure that the variable $\mathsf{H}^\alpha_{\alpha(s)}$ does

not exceed the true goal distance in the abstract transition system for α under the transition cost function $cost^\alpha$. By dualizing the LP without the restriction to alive states in (16.5), we see the connection to flow heuristics. □

On the potential heuristic side, we use abstract states as features with the interpretation that a state s has the feature $s' \in \mathcal{S}^\alpha$ iff $\alpha(s) = s'$. In the special case where α is a projection to k variables, the features $\mathcal{S}^\alpha$ correspond to conjunctions of k atoms. A potential heuristic for a collection of abstractions A then is a consistent and admissible potential heuristic that uses all abstract states of all abstractions in A as features.

Proposition 16.2. *Let Π be a planning task with transitions $\mathcal{T}$, and let A be a set of abstractions for Π with abstract states $\mathcal{S}^\alpha$ for $\alpha \in A$. Let $\mathcal{F}_A = \bigcup_{\alpha \in A} \mathcal{S}^\alpha$ be the set of features for A. Let $C_{\mathcal{F}_A}$ be the following set of constraints.*

$$\sum_{\alpha \in A} \mathsf{W}_{\alpha(s_\star)} \leq 0, \tag{16.6}$$

$$\sum_{\alpha \in A} \mathsf{W}_{\alpha(s')} - \sum_{\alpha \in A} \mathsf{W}_{\alpha(s'')} \leq \mathit{cost}(o) \qquad \text{for all } s' \xrightarrow{o} s'' \in \mathcal{T}. \tag{16.7}$$

The set of solutions of $C_{\mathcal{F}_A}$ is the set of weight functions of admissible and consistent potential heuristic over $\mathcal{F}_A$.

Proof: The important observation for this proof is that we can write the heuristic value of a state $h^w(s) = \sum_{f \in \mathcal{F}_A} w(f)[s \models f]$ as $\sum_{\alpha \in A} w(\alpha(s))$. The constraints then match the definition for goal-awareness (16.6) and consistency (16.7). □

We want to show that $h^{\text{TCP}}_{f,A}$ and an admissible and consistent potential heuristic over features $\mathcal{F}_A$ maximizing the initial heuristic value have the same heuristic value.

Proposition 16.3. *Let s be a state of a planning task Π and A be a set of abstractions of Π. The set of solutions for constraints* (16.1)–(16.3) *(without the restriction to alive states in* (16.2)*) projected to H is equal to the set of solutions f for constraints* (16.6)–(16.7) *with $f(\mathsf{W}_{\alpha(s_\star)}) = 0$ for $\alpha \in A$.*

Proof: Assume f is a solution for constraints (16.1)–(16.3). We show that f' with $f'(\mathsf{W}_{\alpha(s)}) = f(\mathsf{H}^\alpha_{\alpha(s)})$ is a solution to (16.6)–(16.7) with $f'(\mathsf{W}_{\alpha(s_\star)}) = 0$ for $\alpha \in A$. Obviously, constraint (16.1) implies constraint (16.6): if all features that are present in the goal state have a weight of 0, then the total weight of the goal state is 0.

Summing constraint (16.2) over all abstractions for a given transition $s' \xrightarrow{o} s'' \in \mathcal{T}$ shows constraint (16.7):

$$\begin{aligned}\sum_{\alpha \in A} f'(\mathsf{W}_{\alpha(s')}) - \sum_{\alpha \in A} f'(\mathsf{W}_{\alpha(s'')}) &= \sum_{\alpha \in A} f(\mathsf{H}^\alpha_{\alpha(s')}) - \sum_{\alpha \in A} f(\mathsf{H}^\alpha_{\alpha(s'')}) \\ &\overset{(16.2)}{\leq} \sum_{\alpha \in A} f(\mathsf{C}^\alpha_{s' \xrightarrow{o} s''}) \overset{(16.3)}{\leq} \mathit{cost}(o).\end{aligned}$$

For the other direction, let f' be a solution of constraints (16.6)–(16.7) that has $f'(\mathsf{W}_{\alpha(s_\star)}) = 0$ for all $\alpha \in A$, i.e. the weight function of a normalized admissible and consistent potential heuristic. We show that f with $f(\mathsf{H}^\alpha_{\alpha(s)}) = f'(\mathsf{W}_{\alpha(s)})$ and $f(\mathsf{C}^\alpha_{s' \xrightarrow{o} s''}) = f'(\mathsf{W}_{\alpha(s')}) - f'(\mathsf{W}_{\alpha(s'')})$ is a solution for constraints (16.1)–(16.3).

Constraints (16.1) and (16.2) are trivially satisfied for f. Constraint (16.3) is also satisfied, which can be seen by replacing $f(\mathsf{C}^\alpha_{s' \xrightarrow{o} s''})$ by its definition:

$$\sum_{\alpha \in A} f(\mathsf{C}^\alpha_{s \xrightarrow{o} s'}) = \sum_{\alpha \in A} f'(\mathsf{W}_{\alpha(s')}) - \sum_{\alpha \in A} f'(\mathsf{W}_{\alpha(s'')}) \overset{(16.7)}{\leq} \mathit{cost}(o).$$

□

The proposition shows that normalized admissible and consistent potential heuristics correspond to general transition cost partitionings of flow heuristics. Potential heuristics that maximize the value of $h^w(s)$ then have the same value as a general transition cost partitioning that is optimal for s. The normalization (i.e. requiring $w(\alpha(s_\star)) = 0$ for $\alpha \in A$) is not a serious restriction for this argument. As we have seen before for a special case, we can assume that the weight function w is normalized. If this is not the case, consider the normalized weight function w' with $w'(\alpha(s)) = w(\alpha(s)) - w(\alpha(s_\star))$ for all $\alpha \in A$ and $s \in \mathcal{S}$. Let $h^{w'}$ be the induced potential heuristic with $h^{w'}(s) = \sum_{\alpha \in A} w'(\alpha(s)) = h^w(s) - h^w(s_\star)$. As h^w is goal-aware, $h^w(s_\star) \leq 0$, so $h^{w'}$ dominates h^w. The heuristic $h^{w'}$ is still goal-aware and consistent. So, while the restriction to normalized potential functions does exclude some heuristics, these heuristics can never be optimal when we maximize the heuristic value of a single state s.

Theorem 16.2. *Let Π be a planning task and A a set of abstractions for Π. For every state s of Π, let $h^{\text{pot}}_{\mathcal{F}_A, \max s}$ be an admissible and consistent potential heuristic over features $\mathcal{F}_A$ with maximal value for s. Then $h^{\text{pot}}_{\mathcal{F}_A, \max s}(s) = h^{\text{TCP}}_{f,A}(s)$.*

If all abstract states of all transition systems are alive, then $h^{\text{pot}}_{\mathcal{F}_A, \max s}(s) = h^{\text{TCP}}_{h,A}(s)$.

Proof: The statement follows directly from Proposition 16.3 and the fact that the restriction to normalized potential heuristics does not influence the optimal objective value if the heuristic value of a state is maximized. □

The theorem explains the relationship between potential heuristics and general transition cost partitioning for a single state s (the state for which the potential heuristic was optimized). For other states s', we know that $h^{\text{pot}}_{\mathcal{F}_A, \max s}(s') \leq h^{\text{pot}}_{\mathcal{F}_A, \max s'}(s') = h^{\text{TCP}}_{f,A}(s')$ because $h^{\text{TCP}}_{f,A}(s')$ is the value of a potential heuristic that maximizes the heuristic value of s' and $h^{\text{pot}}_{\mathcal{F}_A, \max s}$ thus cannot have a higher value for s'. The heuristic $h^{\text{TCP}}_{f,A}$ finds a new optimal partitioning *for each state*, whereas the potential heuristic uses a weight function that is optimal for a single state s for every heuristic value. What else can we say about $h^{\text{pot}}_{\mathcal{F}_A, \max s}(s')$?

The proof of Proposition 16.3 shows that for a given weight function w, there is a solution f to constraints (16.1)–(16.3) with $f(\mathsf{H}^\alpha_{\alpha(s)}) = w(\alpha(s))$ and $f(\mathsf{C}^\alpha_{s \xrightarrow{o} s'}) =$

$w(\alpha(s)) - w(\alpha(s'))$. The constraints ensure that the transition cost functions $cost_\alpha$ encoded by $\mathbf{C}^\alpha$ form a transition cost partitioning and that $f(\mathsf{H}^\alpha_{\alpha(s)})$ is the abstract goal distance of $\alpha(s)$ in the transition system of α under the cost function $cost_\alpha$. The heuristic value of a state s' then is

$$h^{\text{pot}}_{\mathcal{F}_A,\max s}(s') = \sum_{\alpha\in A} w(\alpha(s')) = \sum_{\alpha\in A} f(\mathsf{H}^\alpha_{\alpha(s')}) = \sum_{\alpha\in A} h^\alpha(s', cost_\alpha).$$

Theorem 16.3. *Let Π be a planning task and $A = \langle\alpha_1, \ldots, \alpha_n\rangle$ a set of abstractions for Π. Let $h^{\text{pot}}_{\mathcal{F}_A,\max s}$ be an admissible and consistent potential heuristic over features $\mathcal{F}_A$ with maximal value for a state s. Then there is a general transition cost partitioning $P = \langle cost_1, \ldots, cost_n\rangle$ for the flow heuristics $f^{\alpha_1}, \ldots, f^{\alpha_n}$ that is optimal for s and has*

$$h^{\text{pot}}_{\mathcal{F}_A,\max s}(s') = f_P(h^{\alpha_1}, \ldots, f^{\alpha_n}, s')$$

for all states s'.

If all abstract states of all transition systems are alive, then the analogous result holds for the abstraction heuristics $h^{\alpha_1}, \ldots, h^{\alpha_n}$.

This answers a question about the relationship of potential heuristics and cost partitioning: admissible and consistent potential heuristics that use abstract states as features compute a general transition cost partitioning of the flow heuristics for the same abstractions. The cost partitioning maximizes the same objective as the potential heuristic (e.g. the heuristic value of the initial state). If no abstraction contains a dead state, the flow heuristics are regular abstraction heuristics. While the heuristics $h^{\text{TCP}}_{f,A}$ and $h^{\text{TCP}}_{h,A}$ would use a separate cost partitioning for every state, potential heuristics optimize the cost partitioning only once and then use this partitioning for all states.

17. Experiments

We now evaluate atomic and binary potential heuristics and compare them to related cost partitioning and operator-counting heuristics from Parts I and II. Our implementation converts the input task to TNF, which does not affect atomic potential heuristics (see Appendix B.3) but can affect binary potential heuristics.

17.1. Atomic Potential Heuristics

We first investigate how much room for improvement there is for atomic potential heuristics with any optimization function. As a baseline for this comparison we use the potential heuristic $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ which optimizes the heuristic value for the initial state. We compare this to the best heuristic that we can aim for, i.e. one that optimizes the potential function for the given state in every evaluation. As shown in Corollary 16.2 this heuristic is equal to the state equation heuristic h^{SEQ}. To measure the relative heuristic quality of $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ and h^{SEQ}, we compare the number of expansions for commonly solved tasks in the first plot of Figure 17.1. We can see that for many tasks the number of expansions for h^{SEQ} is orders of magnitude lower than the one for $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$, which shows that there is still room for improvement for other optimization functions. In total, 747 tasks are solved with $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$, exceeding the coverage of h^{SEQ} by 21. Table 17.1 summarizes the coverage of all considered potential heuristics.

While $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ is guaranteed to match h^{SEQ} on the initial state, it can be arbitrarily off on all other states. In fact, even two instances of $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ can differ on other states because there are usually infinitely many optimal weight functions. The heuristic $h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$ maximizes the average heuristic value of all states (including unreachable states and dead ends) to achieve higher heuristic quality in the state space in general, while possibly sacrificing the high value for the initial state. We deal with dead ends by limiting each feature weight to 10^8. This limit is high enough so dead ends can have heuristic values different from regular states (the ParcPrinter domain has operator costs on the order of 10^6) and also low enough to avoid numerical problems in the LP solver. Overall, this approach solves 32 more tasks than $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ (see Table 17.1). It also has a higher coverage in 24 out of 57 domains, while the opposite is true in only 6 domains. The second plot in Figure 17.1 compares the number of expanded states needed with $h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$ and h^{SEQ}. The overall heuristic quality is better than for $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ but there are also 38 tasks on the y-axis. For such tasks the h^{SEQ} heuristic is perfect in the initial state and needs no expansions to reach the last f-layer, while $h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$ can require up to 10^6 expansions.

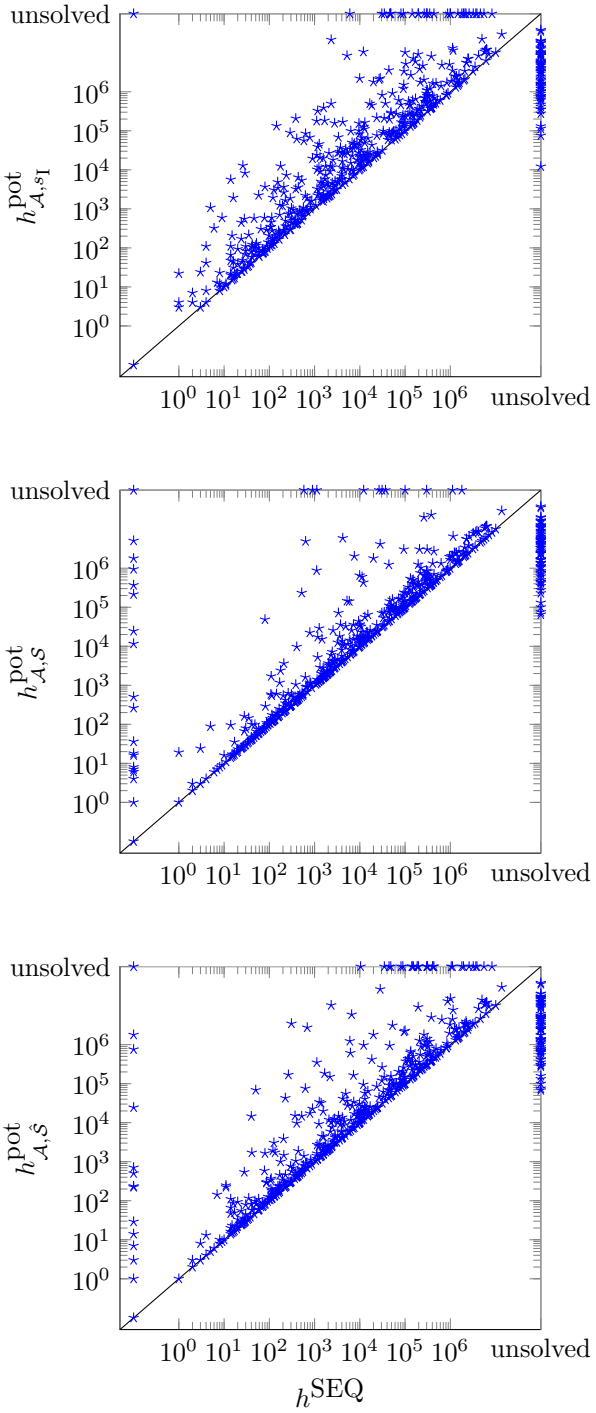


Figure 17.1: Number of expanded states with h^{SEQ} and three potential heuristics.

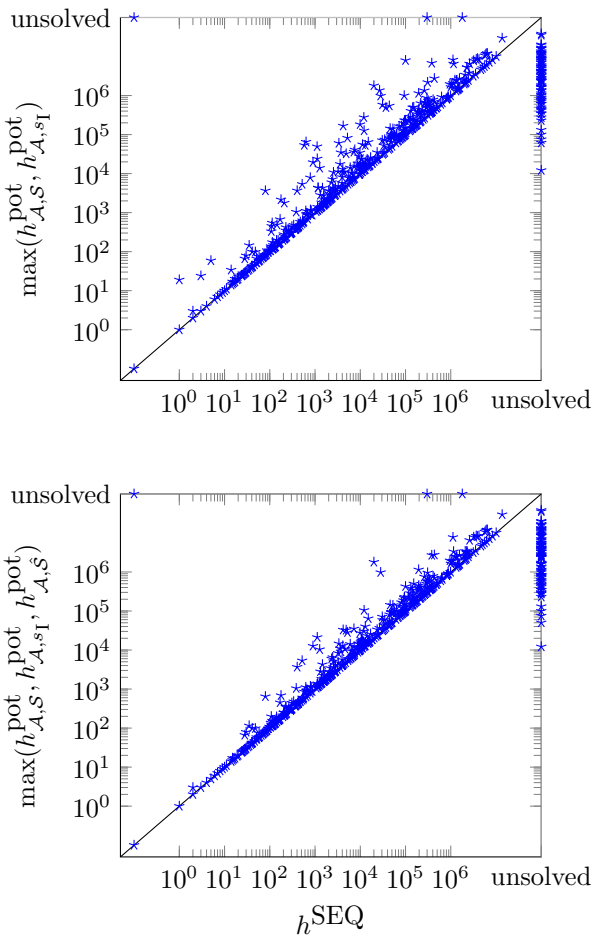


Figure 17.2: Number of expanded states with h^{SEQ} and maxima over several potential heuristics.

Heuristic	Objective	Coverage
$h^{\text{pot}}_{\mathcal{A},s_\text{I}}$	Maximize initial heuristic value	747
$h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$	Maximize heuristic average on all states	779
$h^{\text{pot}}_{\mathcal{A},\hat{\mathcal{S}}}$	Maximize heuristic average on sampled states	746
$\max(h^{\text{pot}}_{\mathcal{A},s_\text{I}}, h^{\text{pot}}_{\mathcal{A},\mathcal{S}})$		806
$\max(h^{\text{pot}}_{\mathcal{A},s_\text{I}}, h^{\text{pot}}_{\mathcal{A},\mathcal{S}}, h^{\text{pot}}_{\mathcal{A},\hat{\mathcal{S}}})$		815
h^{SEQ}	Maximize heuristic value in every state	726

Table 17.1: Number of solved tasks for atomic potential heuristics.

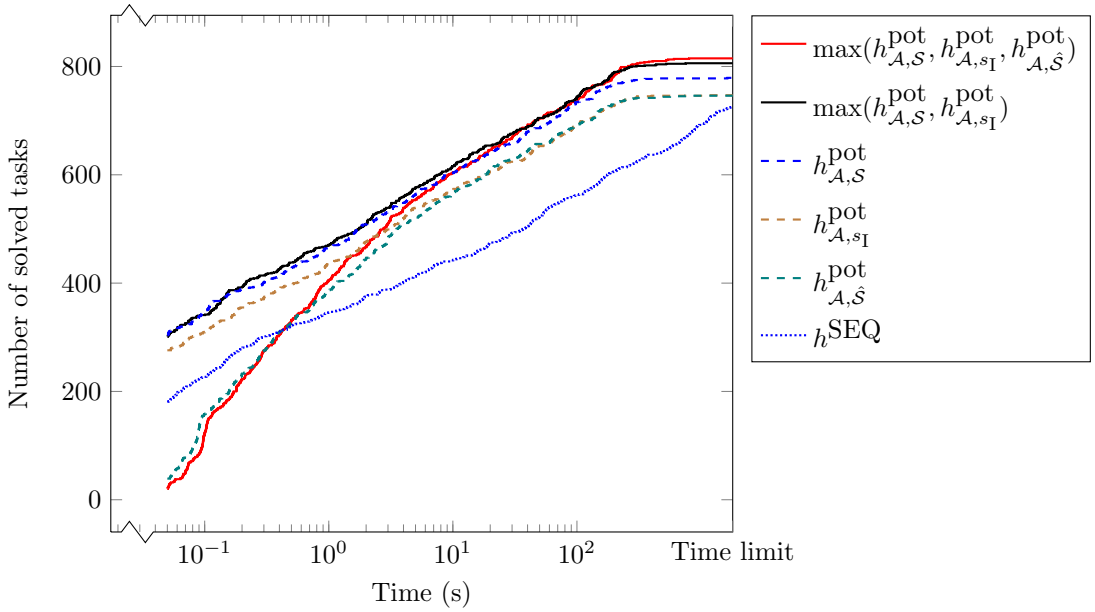


Figure 17.3: Number of solved tasks per time with different potential heuristics.

In addition to focusing the heuristic quality on the initial state ($h^{\text{pot}}_{\mathcal{A},s_\text{I}}$) or distributing it over all states ($h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$), we also try to focus it on the reachable part of the search space. The heuristic $h^{\text{pot}}_{\mathcal{A},\hat{\mathcal{S}}}$ maximizes the average heuristic value of 10^4 reachable states. We collect these samples using random walks with a binomially distributed length centered around the solution depth estimated with $h^{\text{pot}}_{\mathcal{A},s_\text{I}}$. If a random walk gets stuck in a state without successors, it is restarted in the initial state. We limit the weights to 10^8 as in $h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$ to handle the case that some samples are dead ends. The third plot of Figure 17.1 compares the number of expanded states with this heuristic to those needed with h^{SEQ}. The distribution shows similarities with both previous plots. Overall, 746 tasks are solved with $h^{\text{pot}}_{\mathcal{A},\hat{\mathcal{S}}}$, coincidentally almost exactly the number solved with $h^{\text{pot}}_{\mathcal{A},s_\text{I}}$.

Since potential heuristics are fast to compute, we can exploit their complementary strengths by evaluating the maximum of all three heuristics, which solves all tasks solved by at least one of the components. This results in a coverage of 815 tasks, a large improvement over the state equation heuristic which optimizes the cost partitioning for every state and achieves a coverage of 726 (see Table 17.1). Only 11 tasks are solved by h^{SEQ} and not solved by maximizing over the three potential heuristics, while 100 tasks are only solved by the potential heuristics. On the commonly solved tasks the number of states expanded with the maximum over potential heuristics usually is within a factor of 10 of that for h^{SEQ} with 5 exceptions.

The sampling done for $h^{\text{pot}}_{\mathcal{A},\hat{\mathcal{S}}}$ can take a couple of seconds, so we also evaluate the maximum of just $h^{\text{pot}}_{\mathcal{A},s_\text{I}}$ and $h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$. This heuristic also solves every task solved by one of its components and achieves a total coverage of 806 tasks. Figure 17.2 shows the

number of expansions for both combined heuristics and compares them to h^{SEQ}. In both cases, we see how the maximization combines the strengths of its components and brings the heuristic quality closer to that of h^{SEQ}.

The main advantage of potential heuristics is that they are extremely fast to evaluate. This can be seen in Figure 17.3, which compares the number of tasks that could be solved within a given time by the state equation heuristic, optimal cost partitioning heuristics, and the potential heuristics. With the maximum of three potential heuristics 715 tasks are solved in the first minute, compared to 22 minutes to solve 715 tasks for h^{SEQ}. The plot also shows how the sampling process of $h^{\text{pot}}_{\mathcal{A},\hat{\mathcal{S}}}$ makes it less effective in the first seconds of the search. The maximum over $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ and $h^{\text{pot}}_{\mathcal{A},\mathcal{S}}$ is a good alternative for most time limits.

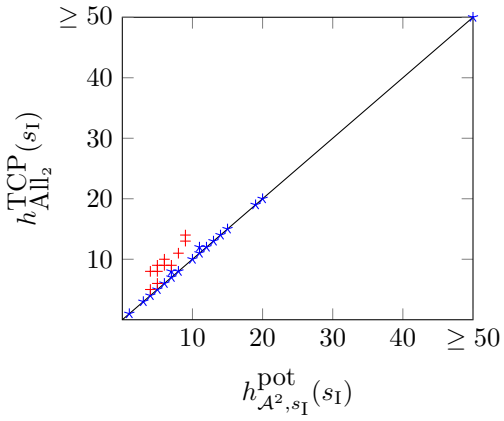
17.2. Binary Potential Heuristics

Binary admissible and consistent potential heuristics correspond to general transition cost partitionings over flow heuristics. The heuristic $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}$ that maximizes the heuristic value of the initial state among such heuristics computes an optimal general transition cost partitioning in the initial state. Compared to h^{TCP} this cost partitioning is not computed in every state, but one that is optimal for the initial state is used for all states.

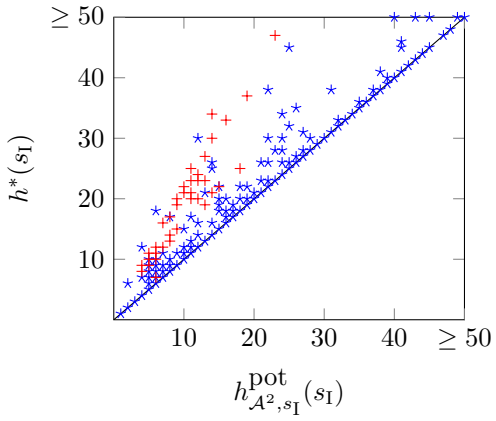
We have already seen in Part I that $h^{\text{TCP}}_{\text{All}_2}$ produces excellent heuristic values, but so far our method of computing them was exponential in the task's description size. With $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}$ we have an alternative that partitions the cost over heuristics that minimize flow instead of the the cost of the shortest path. Figure 17.4a compares initial heuristic values of these two heuristics. There are only few domains where the difference between minimizing flow and path cost makes a difference. Most notable is the domain PSR which is highlighted in the plot.

We can compute the weights of $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}$ in polynomial time, even though this is still expensive. With this cheaper method of computation we can now see the heuristic accuracy in the initial state for a wider range of tasks. In 346 out of 529 cases (53%) where we could determine both $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}(s_{\text{I}})$ and $h^*(s_{\text{I}})$, the $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}$ value is perfect. These perfect values occur in 39 of the 57 domains. The difference between $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}$ and h^* is shown in Figure 17.4b. There are still some heuristic values that are far from the optimum. In the domain PSR the heuristic is inaccurate because it minimizes flow instead of path cost but there are suboptimal values in other domains as well, suggesting that using features of size three or larger might be necessary in some domains.

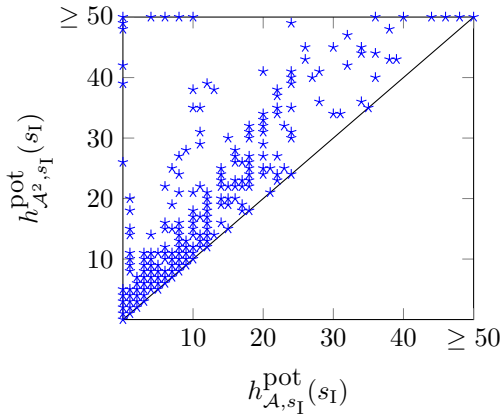
Figure 17.4c compares initial heuristic values of $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ and $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}$. As expected, larger features frequently achieve higher heuristic values. The initial heuristic value also is perfect less frequently. Out of the 529 tasks mentioned above, only 104 (20%) have a perfect $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}$ value. There are also several cases in which $h^{\text{pot}}_{\mathcal{A},s_{\text{I}}}(s_{\text{I}})$ is 0, while $h^{\text{pot}}_{\mathcal{A}^2,s_{\text{I}}}$ reports values as high as 66.



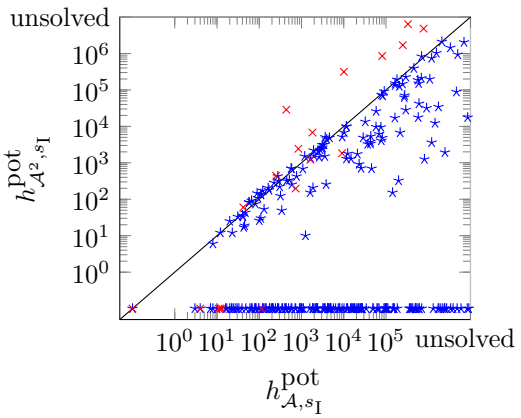
(a) Initial heuristic values of $h^{\mathrm{pot}}_{\mathcal{A}^2,s_{\mathrm{I}}}$ and $h^{\mathrm{TCP}}_{\mathrm{All}_2}$. The domain PSR is highlighted.



(b) Initial heuristic values of $h^{\mathrm{pot}}_{\mathcal{A}^2,s_{\mathrm{I}}}$ and h^*. The domain PSR is highlighted.



(c) Initial heuristic values of $h^{\mathrm{pot}}_{\mathcal{A},s_{\mathrm{I}}}$ and $h^{\mathrm{pot}}_{\mathcal{A}^2,s_{\mathrm{I}}}$.



(d) Expansions of $h^{\mathrm{pot}}_{\mathcal{A},s_{\mathrm{I}}}$ and $h^{\mathrm{pot}}_{\mathcal{A}^2,s_{\mathrm{I}}}$. The domain Blocksworld is highlighted.

Figure 17.4: Heuristic quality binary potential heuristics compared to atomic potential heuristics, optimal transition cost partitioned abstraction heuristics, and the optimal heuristic.

If atomic and binary potential heuristics are optimized for a maximal initial heuristic value, the value for the binary potential heuristic is at least as high as the one for the atomic one. The heuristic values of other states, however, can be arbitrarily different. To compare the overall quality of the two heuristics, Figure 17.4d shows the number of expansions in an A* search. Using $h^{\text{pot}}_{\mathcal{A}^2,s_\text{I}}$ almost always leads to fewer expansions than $h^{\text{pot}}_{\mathcal{A},s_\text{I}}$. The exceptions are mostly from the domain Blocksworld. The tasks on the x-axis of the plot are those where $h^{\text{pot}}_{\mathcal{A}^2,s_\text{I}}$ is perfect.

18. Related and Future Work

We finish the topic of potential functions by discussing two applications where they are not used as admissible heuristics. We also discuss related work in multi-agent planning and the problem of finding good feature sets.

18.1. Correlation Complexity

So far, we have considered optimal planning, where discovered plans must have minimal cost. In *satisficing planning* we only care about finding any plan independent of its cost. The problem of deciding whether any plan exists is still PSPACE-complete in general, but many planning domains are known to admit polynomial domain-specific solution algorithms (e.g. Helmert, 2003). Perhaps "simple" potential heuristics only considering conjunctions of 2 or 3 atoms are already highly accurate in these "simple" domains?

Unfortunately, there is bad news in the literature: Helmert and Mattmüller (2008) showed that h^m and (single) PDB heuristics based on conjunctions of bounded size give rise to arbitrarily bad heuristics in all domains they studied. However, they also showed that additive heuristics based on multiple PDBs can be significantly more accurate. This is not just good news for PDBs but also for potential heuristics, which are additive combinations of simpler heuristics by definition.

So just how complex does a potential heuristic have to be so that solving a planning task becomes simple? Following in Hoffmann's (2005) footsteps, we formalize this question by considering *per-domain* results for the *state space topology* of planning tasks. Hoffmann studied the search space topology of a fixed heuristic, namely the optimal delete relaxation heuristic h^+. Delete relaxation heuristics are rarely perfect but frequently good: to quantify this, Hoffmann focused on the size of local minima in the state space to distinguish "easy" from "difficult" domains for h^+.

In contrast, potential heuristics can be as accurate as we wish, at a cost in heuristic complexity. To reflect this degree of control, in our theoretical analysis we are more demanding with state space topology, looking for heuristics that exhibit *no local minima at all*. The question, then, is how complex a potential heuristic needs to be in order to have no local minima.[1] We measure this complexity in the size of the conjunctions required, i.e. in the dimension of the potential heuristic.

[1] By "local minimum" we mean any state which does not have a successor with lower heuristic value. This includes states within heuristic plateaus.

To formalize heuristics that exhibit no local minima, we must first clarify what we mean by that. A tentative definition might be: "every non-goal state has a successor with lower heuristic value". However, this is too strict: in a finite state space, such a definition implies that there is a strictly descending path towards a goal state from every state, which is impossible to satisfy if the task has any unsolvable states.

Hence, we only require that *solvable* states have successors with lower heuristic value. To stop a heuristic search algorithm from getting trapped in an unsolvable region of the state space, we also require that unsolvable successors s' of a solvable state s never have a lower heuristic value than s.

A second problem is that planning tasks often include "impossible" states that violate physical constraints, such as two blocks being stacked on top of each other in the Blocksworld domain. It would be unnecessarily restrictive to require that the state space topology is also well-behaved for such impossible states. However, there is in general no simple way to distinguish possible from impossible states without complicating the definition of planning tasks. A simple remedy is to restrict attention to *reachable* states.

The solvable and reachable states are exactly the states that are alive. With the restriction to such states, we can now introduce two criteria that together imply absence of local minima.

Definition 18.1 (descending)**.** *A heuristic h is* descending *if all alive, non-goal states have an improving successor, i.e. for every such state s there is an applicable operator o with $h(s[\![o]\!]) < h(s)$.*

Definition 18.2 (avoiding dead ends)**.** *A heuristic* avoids dead ends *if all improving successors of alive, non-goal states are solvable.*

Given these two properties typical heuristic search algorithms for satisficing planning are guided directly towards the goal. Seipp et al. (2016a) give a formal proof for simple hill-climbing. The main idea is that the heuristic value of the currently considered state must decrease in every expansion.

Theorem 18.1. *Let h be a descending, dead-end avoiding heuristic for a planning task Π with state $\mathcal{S}$. Further let $d = \min_{s,s' \in \mathcal{S}, h(s) \neq h(s')} |h(s) - h(s')|$ be the smallest non-zero heuristic difference between two states and let $L = (h(s_\mathrm{I}) - \min_{s \in \mathcal{S}} h(s))/d$.*

Then simple hill-climbing with h solves Π after at most L state expansions if Π is solvable and returns with failure after at most L state expansions if Π is unsolvable.

Note that in the common case where heuristic values are non-negative integers, the bound L simplifies to at most $h(s_\mathrm{I})$. Any descending dead-end avoiding potential function with rational weights can be turned into a heuristic function with non-negative integer values by multiplying all weights with their lowest common denominator and then adding a constant.

We can now formally define a complexity measure for a planning task: *correlation complexity* measures how complex a potential heuristic must be to obtain a favorable state space topology.

Definition 18.3 (correlation complexity of a planning task)**.** *The* correlation complexity *of a planning task* Π *is the minimum dimension* d *of all descending, dead-end avoiding potential heuristics for* Π*.*

The correlation complexity of a planning task is bounded from above by the number of state variables. To see this, consider the heuristic d^* that maps every state to the smallest number of operators in any s-plan or to ∞ is no s-plan exists. Let D be the largest finite value of d^* and let h be the heuristic d^* with all infinite values replaced by $D + 1$. The heuristic h is descending since every alive non-goal state has a successor along a shortest s-plan which has a lower value. It is also dead-end avoiding since all unsolvable states have the value $D + 1$, so they cannot be improving successors of any other state. As mentioned in Chapter 14, a finite heuristic function like h can be seen as a potential function of dimension $|\mathcal{V}|$, so h demonstrates that the correlation complexity of the given task cannot exceed $|\mathcal{V}|$. In particular, this guarantees that the correlation complexity of planning tasks is well-defined.

The definition can be extended to *planning domains*, which for the purposes of this thesis are simply (usually infinite) sets of planning tasks.

Definition 18.4 (correlation complexity of a planning domain)**.** *The* correlation complexity *of a planning domain is the maximal correlation complexity of all planning tasks in the domain, or* ∞ *if no maximum exists.*

If a domain has low correlation complexity, this is a sign that no complex interactions between variables need to be considered in order to solve planning tasks in this domain. Hence, low correlation complexity is an indication that a planning domain is "easy".

A formal tractability result for planning in such a domain does not immediately follow because Definition 18.4 does not guarantee that a low-dimension potential heuristic for a given planning task is easy to construct – it only guarantees that such a potential heuristic exists. Moreover, planning domains with low correlation complexity can have exponentially long plans. For example, it is easy to construct "binary counter" tasks $(\Theta_i)_{i\geq 1}$ (as we did in Section 5.1) with correlation complexity 1 where Θ_i requires plans of length 2^i to solve. In the absence of such complications, low correlation complexity indeed implies tractability.

Theorem 18.2. *Let* $\mathcal{D}$ *be a planning domain with correlation complexity* $d < \infty$*, and let* p *be a polynomial such that given* $\Pi \in \mathcal{D}$ *with encoding size* n*,*

1. *a descending, dead-end avoiding potential heuristic* φ_Π *of dimension* d *can be computed in time* $p(n)$*, and*
2. *feature weights are integers and polynomially bounded:* $|w(F)| \leq p(n)$ *for all features* F *of* φ_Π*.*

Then plan generation in $\mathcal{D}$ *can be solved in polynomial time.*

Proof: A task with encoding size n has at most n state variables, and hence φ_Π has no more than $O(n^d)$ features. Together with the bound on the individual weights, it follows that $|\varphi_\Pi(s)| \leq O(n^d)p(n)$ for all states s, and hence the difference between the heuristic values of any two states is bounded by a polynomial in n.

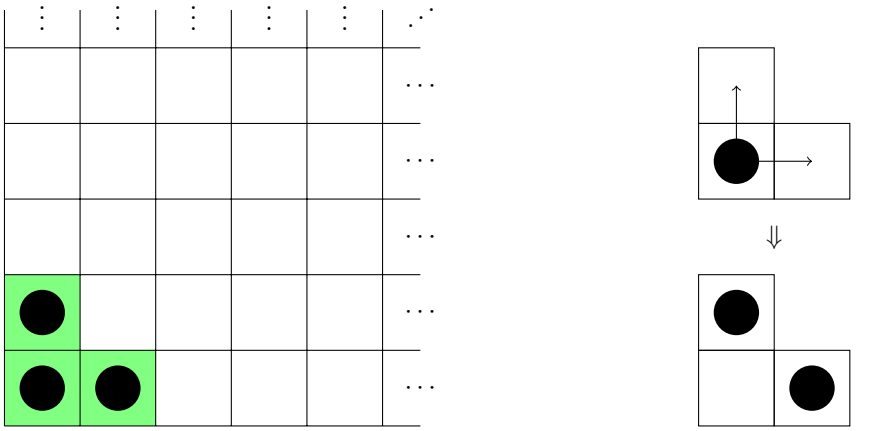
The result follows with Theorem 18.1, as heuristic values are integers, L is thus bounded by a polynomial in n, and each heuristic evaluation can be performed in time $O(n^d)$, which is also polynomial in n. □

Seipp et al. (2016a) study the correlation complexity for a number of well-known planning domains. It turns out that most domains they investigated (Spanner, Gripper, VisitAll and Blocksworld) have a low correlation complexity of 2. Given that potential heuristics with low dimension can be evaluated very efficiently, these results motivate further research on how to find good features and weights for low-dimensional potential heuristics automatically. For optimal planning, we have also seen that finding good weights for binary potential heuristics is fundamentally easier then finding them for ternary features. While this result does not necessarily carry over to satisficing planning, it is encouraging to see that binary features are often sufficient.

At its core, correlation complexity describes how tightly interrelated different aspects of a planning task are, and to what extent these aspects can be considered separately. The general idea of measuring the degree of interrelatedness between state variables of a planning task is not new. In a line of research with very similar motivations, Chen and Giménez (2007; 2009) studied several notions of *width* for planning tasks, where low width implies low complexity of planning. In the same spirit, Lipovetzky and Geffner (2012) also introduced a notion of *width* (different from those of Chen and Giménez) and exploited it to efficiently solve a large number of standard planning benchmarks. All definitions of width are incomparable to correlation complexity, meaning that there exist task with high correlation complexity and low width, and vice versa (Seipp et al., 2016a). This is to be expected: in both cases, the intuition is that low complexity means that solutions can be found efficiently. For example, a solution to a task with low width can be found efficiently with the iterated width algorithm (Lipovetzky and Geffner, 2012). In a problem with low correlation complexity, there exists a potential function with which a solution can be found without backtracking (with the added difficulty that low complexity only means that such a function exist, but does not tell us how to construct it). The converse is not necessarily true: if a domain has high width (for example), this does not imply that planning is hard in this domain, only that an algorithm based on exploiting low width might not be suitable for it.

18.2. Dead-end Detection

We start this section with a puzzle called "Pebbling the chessboard" (Kontsevitch, 1981; Chung et al., 1995) or "Freeing the clones" (Numberphile, 2013). Consider the infinite grid shown in Figure 18.1a. The three marked cells in the corner are called the *prison*



(a) Initial situation **(b)** Movement rule

Figure 18.1: Clone escape puzzle

and initially it contains three imprisoned *clones*. A clone can only move if the cell above it and to its right are both empty. If this is the case, the clone can duplicate, moving one copy up and the other to the right as shown in Figure 18.1b. The goal is to find a sequence of moves that allows all clones to escape the prison, i.e. one that ends in a situation where all three prison cells are empty. We invite the reader to take some time to try and find a solution. We will discuss it later in this section.

In the above puzzle it is important to recognize dead ends, states from which no goal state is reachable. This is useful for planning tasks in general for two reasons: when a search is used to solve a task, all recognized dead ends can be pruned, which can significantly reduce the size of the search space. Additionally, the initial state is a dead end iff the task is unsolvable. The problem of deciding whether a task is solvable recently gained a lot of attention (e.g. Bäckström, Jonsson, and Ståhlberg, 2013; Hoffmann, Kissmann, and Torralba, 2014; Lipovetzky, Muise, and Geffner, 2016). Here, we show how potential functions can be used to identify a state as a dead end. We assume that planning tasks have a unique goal state (as for example in TNF) but the technique can easily be extended to unrestricted SAS^+ tasks.

An important concept for recognizing dead ends are *invariants* (e.g. Rintanen, 2000; Gerevini and Schubert, 2001; Helmert, 2009; Alcázar and Torralba, 2015; Rintanen, 2017), properties that are maintained through operator application.

Definition 18.5 (invariant)**.** *Let Π be a planning task with transitions $\mathcal{T}$. A property P of states of Π is an* invariant *if $P(s)$ implies $P(s')$ for all transitions $s \xrightarrow{o} s' \in \mathcal{T}$.*

If an invariant holds in a state s, it also must hold in all states reachable from s. If it additionally does not hold in $s_\star$, then there can be no path from s to $s_\star$. In the following, we use the initial state in definitions and results instead of an arbitrary state s. This is no

limitation because all techniques can be used for any state s by treating s as the initial state.

Proposition 18.1. *Let Π be a planning task with initial state s_{I} and unique goal state $s_\star$. Let P be an invariant of Π. If P holds in s_{I} but not in $s_\star$, then Π is unsolvable.*

Proof: If Π were solvable, then P would have to hold in all states visited by a plan, including $s_\star$. □

We can use potential functions to describe invariants. This is not a restriction, as any function that maps states to numbers can be expressed with a potential function over sufficiently high-dimensional features. For a given potential function φ, a state s has the property $P_{\varphi,c}$ iff $\varphi(s) \geq c$. This property is an invariant if $\varphi(s) \geq c$ implies $\varphi(s') \geq c$ for all transitions $s \xrightarrow{o} s' \in \mathcal{T}$. If the potential of a state can never decrease ($\varphi(s) \leq \varphi(s')$), then this potential function defines an invariant. If additionally the initial state has a higher potential than the goal state (i.e. $\varphi(s_{\mathrm{I}}) \geq c > \varphi(s_\star)$), the task cannot be solvable. Every operator application can only further increase the potential, so there is no way to reach the goal state where the potential must be lower than in the initial state. We call functions with this property *separating functions* because they separate the goal state from the set of reachable states.

Definition 18.6 (separating function)**.** *Let Π be a planning task in TNF with initial state s_{I}, goal state $s_\star$ and transitions $\mathcal{T}$. A* separating function *is a function φ that maps states of Π to numbers and satisfies*

$$\begin{aligned} &\varphi(s_{\mathrm{I}}) > \varphi(s_\star) \quad \textit{and} \\ &\varphi(s) \leq \varphi(s') \quad \textit{for all } s \xrightarrow{o} s' \in \mathcal{T}. \end{aligned}$$

Proposition 18.2. *A planning task in TNF has a separating function iff it is unsolvable.*

Proof: For the "if" part consider the function φ for an unsolvable task Π that maps every reachable state to 1 and every unreachable state to 0. There are no transitions from reachable to unreachable state, so the potential can never decrease along a transition. The initial state is always reachable, and since Π is unsolvable, the goal state is not reachable, so $\varphi(s_\star) = 0 < 1 = \varphi(s_{\mathrm{I}})$. Therefore, φ is a separating function if Π is unsolvable.

For the "only if" part it is easy to see that $P_{\varphi,\varphi(s_{\mathrm{I}})}$ is an invariant that holds in the initial state but not in the goal state. Proposition 18.1 then shows the statement. □

A separating function can be used in the example from the beginning of the section to show that there is no way for the clones to escape the prison. Figure 18.2 assigns a number to each cell, which can be interpreted as the weights w of a potential function φ. The potential of a given state is the sum of weights for all occupied cells. The weight of cell $\langle i, j\rangle$ is $w(\langle i, j\rangle) = 2^{-(i+j)}$. A move from cell $\langle i, j\rangle$ makes the occupied cell $\langle i, j\rangle$ empty and the two empty cells $\langle i + 1, j\rangle$ and $\langle i, j + 1\rangle$ occupied. The potential

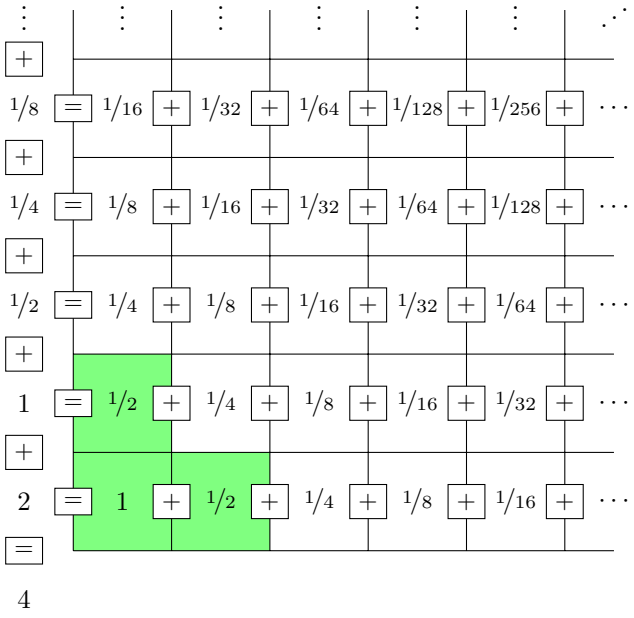


Figure 18.2: Weight function for clone escape puzzle

thus does not change because $-2^{-(i+j)} + 2^{-(i+1+j)} + 2^{-(i+j+1)} = 0$. In the initial state, the three prison cells are occupied for a total potential of $\varphi(s_\mathrm{I}) = 1 + 1/2 + 1/2 = 2$. To see the maximal possible potential of a goal state, we first compute the sum of all cell weights. As shown next to the board in Figure 18.2, the sum of all weights in row j is $\sum_{i\geq 0} 2^{-(i+j)} = 2^{-j} \sum_{i\geq 0} 2^{-i} = 2^{1-j}$ and the sum over all rows is $\sum_{j\geq 0} 2^{1-j} = 2\sum_{j\geq 0} 2^{-j} = 4$. Since the prison cells have a total weight of 2, all remaining cell weights also sum up to 2. In a finite number of moves it is not possible to cover an infinite number of cells, so all goal states $s_\star$ have $\varphi(s_\star) < 2 = \varphi(s_\mathrm{I})$. As the potential of all reachable states is 2, no goal state can be reachable.

Many other tasks can be shown to be unsolvable with a similar argument. For example, the "mutilated chessboard problem" (Black, 1946) asks if there is a way to place 31 2×1 dominoes on a chess board that has two opposing corners broken off. McCarthy (1964) conjectured that this problem is particularly difficult for automated proof procedures even though an informal proof is simple: each domino covers a black and a white square, and since the two missing corners had the same color, there are more squares of one color than of the other. Separating functions mirror this argument directly: counting 1 for each covered black square and -1 for each covered white square (see Figure 18.3). The potential is initially 0 and not changed by any transition (placing a domino covers a black and a white square for a total change of $1 - 1 = 0$). In the goal state, all squares are covered, so the potential is -2. This potential function has the property of a separating function and thus shows unsolvability.

Proposition 18.2 guarantees that there is a separating function for unsolvable ev-

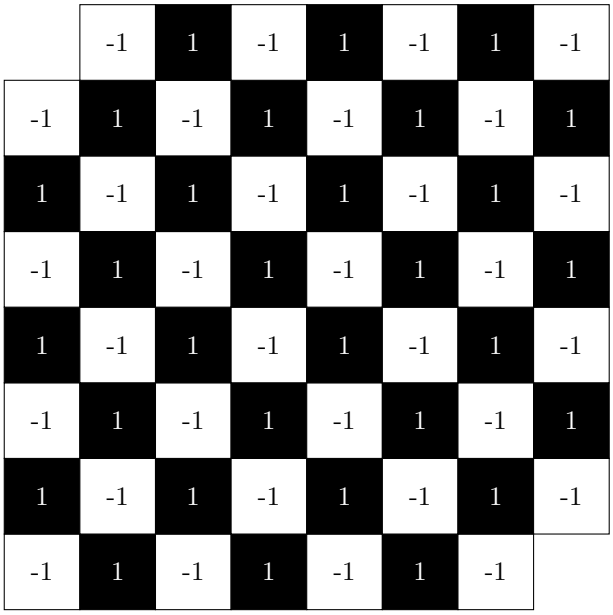


Figure 18.3: Weight function for mutilated chessboard puzzle

ery task. However, this function might require high-dimensional features. Ideally, we would like to use an LP solver to *synthesize* a separating function for a given set of features $\mathcal{F}$. If a solution to the constraints in Definition 18.6 exists, then the task is unsolvable. There are two problems with this: the number of constraints is exponential in the task description size because there is one constraint for each transition, and the constraints contain a strict inequality. LPs with strict inequalities are not guaranteed to have a well-defined optimal objective value, as for example maximizing x subject to $x < 1$. In the case of separating functions, there are several ways to deal with this. We can require a separation of at least 1 (with $\varphi(s_\mathrm{I}) \geq 1 + \varphi(s_\star)$), but there is an option that that is more useful for the following discussion. The constraint $\varphi(s_\mathrm{I}) > \varphi(s_\star)$ is satisfied iff there is a number c such that $\varphi(s_\mathrm{I}) > c$ and $c \geq \varphi(s_\star)$. Instead of testing whether a solution of these constraints exists, we can ignore the $>$-constraint, maximize $\varphi(s_\mathrm{I})$ subject to the remaining constraints, and check whether the resulting optimal objective value is larger than c. It is sufficient to limit our attention to the case $c = 0$ if $\mathcal{F}$ contains the empty feature $\top$ because in this case every separation function can be "shifted" by $-\varphi(s_\star)$ to satisfy the constraints. In general, such a shift is possible whenever $\mathcal{F}$ contains a set of features of which exactly one is true in every state, e.g. a feature for every atom of a variable. The resulting LP then is:

$$\begin{aligned} &\text{Maximize } \varphi(s_\mathrm{I}) \text{ subject to} \\ \varphi(s_\star) &\leq 0 \\ \varphi(s) - \varphi(s') &\leq 0 \qquad\qquad \text{for all } s \xrightarrow{o} s' \in \mathcal{T}. \end{aligned}$$

Note that the constraints of the above LP encode exactly goal-awareness and consistency of φ in the task Π with the cost function $\mathit{cost}(o) = 0$ for all operators o. Ignoring the operator costs in this way makes sense because they do not influence whether the task is solvable. This offers a new view on the argument for unsolvability: if there is an admissible heuristic function for the task that ignores all costs and this heuristic function has a positive (non-zero) value for the initial state, then the task cannot be solvable. Under such a cost function, all plans are free of cost, so a positive heuristic value for s_{I} can only be admissible if there is no plan.

The LP still contains an exponential number of constraints, but we already discussed how and when admissible and consistent potential functions can be characterized compactly (see Chapter 15).

Aidos, the winner of the unsolvability IPC (Muise and Lipovetzky, 2014), is a portfolio of planning techniques including one based on separating functions (Seipp et al., 2016b). It uses binary features but weaker constraints compared to ones described in Section 15.2. Seipp et al. omit features f with $s_\star \models f$, which normalizes the potential of the goal state to 0 and makes the constraint $\varphi(s_\star) \leq 0$ redundant. They also restrict all weights to non-negative numbers. This excludes some solutions, which means that there might be fewer tasks for which the planner finds a separating function. Tasks where a separating function with non-negative weights exists are still provably unsolvable, though.

Considering only non-negative weights leads to the following lower bound estimation that weakens the method further but reduces the number of necessary constraints in the LP. We extend the notation from Section 9.4 and say that an operator o *consumes* a feature f in a state s if $s \models f$ and $s[\![o]\!] \not\models f$. We say o *can consume* f if there is a state s in which o is applicable and consumes f. Analogously, o *produces* f in a state s if $s \not\models f$ and $s[\![o]\!] \models f$ and we say that o *always produces* f if o produces f in all states in which it is applicable. Whether an operator can (or is guaranteed to) consume or produce a feature can easily be checked syntactically. Since all weights are non-negative, we can then bound the potential difference along a specific transition $s \xrightarrow{o} s' \in \mathcal{T}$ as follows:

$$\begin{aligned}
\varphi(s) - \varphi(s') &= \sum_{f \in \mathcal{F}} w(f)[s \models f] - \sum_{f \in \mathcal{F}} w(f)[s' \models f] \\
&= \sum_{\substack{f \in \mathcal{F} \\ o \text{ consumes } f \text{ in } s}} w(f) \; - \sum_{\substack{f \in \mathcal{F} \\ o \text{ produces } f \text{ in } s}} w(f) \\
&\leq \sum_{\substack{f \in \mathcal{F} \\ o \text{ can consume } f}} w(f) \; - \sum_{\substack{f \in \mathcal{F} \\ o \text{ always produces } f}} w(f) = \Delta_o(f)
\end{aligned}$$

The expression $\Delta_o(f)$ no longer depends on the state s and since it is an upper bound for $\varphi(s) - \varphi(s')$, $\Delta_o(f) \leq 0$ implies $\varphi(s) - \varphi(s') \leq 0$.

Putting things together, a task is unsolvable if the set of features $\mathcal{F}$ contains no feature that is present in the goal state, and the following LP has an optimal objective value higher than 0:

$$\text{Maximize} \sum_{f \in \mathcal{F}} \mathsf{W}_f[s \models f] \text{ subject to}$$

$$\sum_{\substack{f \in \mathcal{F} \\ o \text{ can consume } f}} \mathsf{W}_f \; - \sum_{\substack{f \in \mathcal{F} \\ o \text{ always produces } f}} \mathsf{W}_f \leq 0 \qquad \text{for all } o \in \mathcal{O}$$

$$\mathsf{W}_f \geq 0 \qquad \text{for all } f \in \mathcal{F}$$

As another optimization, note that if a feature is not always produced by at least one operator, then its weight only occurs with a coefficient of 1 and setting it to 0 cannot make a satisfied constraint unsatisfied. Aidos therefore does not consider such features. In an evaluation after the IPC, Seipp et al. identified this technique as the most influential component of their portfolio in 7 out of 15 domains.

Aidos also has a second component that can be understood as a separating potential function (Pommerening and Seipp, 2016). It iteratively creates larger and larger PDBs and extracts their reachable, dead states as partial states. Any state consistent with one of these partial states must be dead and can be pruned in a search.

To view this as a separating function, we have to generalize the definition slightly: we say a function φ is a separating function *for a state* s if $\varphi(s) > \varphi(s_\star)$ and $\varphi(s') \leq \varphi(s'')$ holds for all transitions $s' \xrightarrow{o} s''$ between *alive* states. There is a separating function for a state s iff there is no s-plan. (The restriction to transitions between alive states does not affect Proposition 18.2 because plans can only use such transitions.) Consider a set of features $\mathcal{F}_\text{D}$, where every feature corresponds to a reachable, dead state in an abstraction and a weight function w that assigns a positive value to all of them. The potential function φ for $\mathcal{F}_\text{D}$ and w only has a positive value for reachable, dead states. This implies $\varphi(s_\star) = 0$ and $\varphi(s') = 0 = \varphi(s'')$ for all transitions $s' \xrightarrow{o} s''$ between alive states. If $\varphi(s) > 0$, then φ is a separating function for s and there is no s-plan.

If $\mathcal{F}_\text{D}$ contains all reachable, dead states from the projections in Int_k, it can be used to test if one of the projections would detect a given state as unsolvable without storing all pattern databases. This corresponds to testing for k-consistency (Bäckström, Jonsson, and Ståhlberg, 2013) in every state.

18.3. Multi-Agent Planning

Multi-Agent Planning (Brafman and Domshlak, 2008) differs from classical planning by considering multiple cooperating agents. Each agent has private and public capabilities, represented by private and public state variables and operators. The public capabilities of all agents are shared while the private ones should only be known to the

agent. Distributed planning algorithms such as MAD-A* (Nissim and Brafman, 2012) are able to find globally optimal solutions while preserving the privacy of the agents under certain additional assumptions. Each agent plans locally with its private operators and the set of all global operators. Coming up with admissible heuristics is complicated in this framework as no agent knows the full planning task. Each agent can compute an admissible heuristic for its view of the task, but these estimates are not necessarily additive.

Štolba, Fišer, and Komenda (2016) use potential heuristics to derive admissible estimates for each agent such that the heuristics computed by each agent are additive. They use the set of all atoms as features and partition it into one set for all public atoms and one set for each agent. Admissible potential heuristics then can be seen as a collection of additive potential heuristics for each agent. (The public part can be handled by one of the agents.) Such heuristics can be evaluated fast and in a distributed way.

Štolba, Fišer, and Komenda discuss three ways of computing the weights for the potential functions. All of them aim to optimize the weights subject to the constraints for admissibility and consistency. The easiest option is to designate one agent to collect the complete task and solve the LP but this does not preserve privacy. A second option uses Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) as suggested by Holmgren, Persson, and Davidsson (2009) to split up the LP among the agents. This also has no formal privacy guarantees but avoids openly sharing the definitions of private operators. The third option they suggest keeps the information of the agents' parts in the LP private by computing the LP with privacy-preserving distributed methods (Mangasarian, 2011; Dreier and Kerschbaum, 2011).

Using the potential heuristic in MAD-A* achieves state-of-the-art performance and outperforms a distributed version of the LM-cut heuristic (Fišer, Štolba, and Komenda, 2015).

18.4. Finding Good Feature Sets

To close the part on potential heuristics, we discuss a possible future research direction. In Chapter 15 we introduced methods to find good weights for admissible potential heuristics over a given set of features but we left open how to select a good sets of features. Using all atomic and/or binary features is a good starting point and finding admissible potential heuristics for them remains polynomial. Empirically, using binary features often leads to significantly stronger heuristics than just using atomic features but their weight functions can take a long time to compute. But not all features are needed to achieve this heuristic quality.

Figure 18.4 shows that often many features have a zero weight in a weight function where the initial heuristic value is maximal. (The distribution is similar for other objective functions.) Every feature with a weight of zero could be removed without affecting the heuristic value. For atomic features, there are even a few cases where all

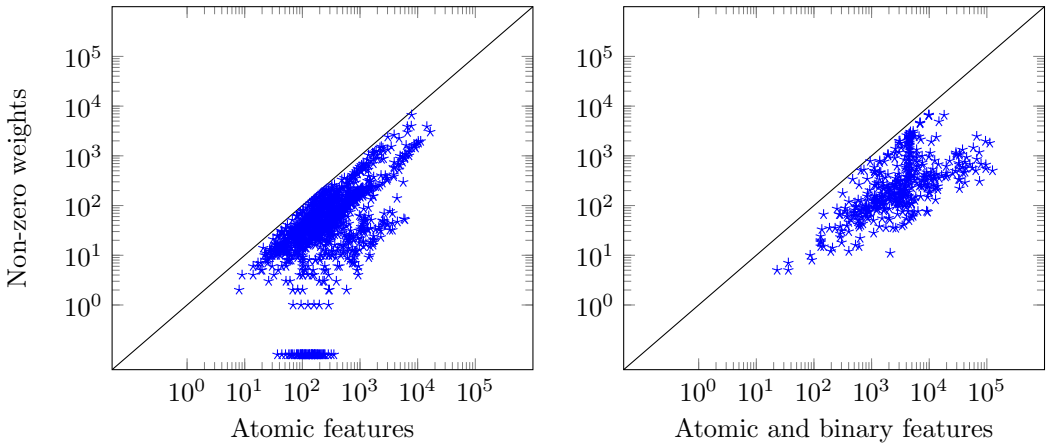


Figure 18.4: Number of features compared to the number of features with a non-zero weight in a potential heuristic with maximal value for the initial state.

weights are zero because no admissible and consistent potential heuristic is better than the blind heuristic, which is possible in the presence of zero-cost operators. With binary features, there is always at least one feature with a positive weight but the majority of weight functions still use only a fraction of the features. This demonstrates that there is a lot of room for improvement: by selecting the right features high quality heuristics could be synthesized in a reasonable time.

There are different approaches to selecting features. Bonet and van den Briel (2014) describe an iterative procedure to add values of partially merged variables to the state equation heuristic which we discussed in Section 11.1.5. As we have seen, the state equation heuristic and its extension for partial merges are cost-partitioned flow heuristics, which in turn correspond to a potential heuristic. The iterative procedure can thus be seen as a technique to find feature sets. Bonet and van den Briel add new features if an operator has a non-zero count in a solution. To interpret this in the context of potential heuristics, we remark that an LP variable is non-zero in an optimal solution iff the corresponding dual constraint is tight, i.e. its left and right hand sides are equal. This is called *complementary slackness* and is a consequence of the duality theorem (Schrijver, 1998). Looking for operators with a non-zero count in an optimal state equation heuristic solution is thus equivalent to looking for tight constraints in an optimal potential heuristic solution. Such a tight constraint represents an operator o with at least one transition where the heuristic difference exactly matches the operator costs. Following the partial merge strategy, the newly created feature mentions at least one prevail and one non-prevail precondition. Such a feature is guaranteed to be consumed by o and thus occurs in the tight constraint. Like the partial merge strategy, this strategy could be run once before starting the search until a fixed point is reached.

An alternative would be to learn features during the search. The weight function of a potential heuristic can be updated during a search, either by re-optimizing the weights

for a different objective or by adding new features and finding good weights for them. Since potential functions are fast to evaluate, it might also be useful to add new potential functions instead of replacing the existing one. An interesting direction for future research would be to identify heuristic errors during the search and add features or potential functions that prevent errors of this kind in the rest of the search. The approach by Bonet and van den Briel can be seen as a step in this direction as it prevents the errors caused by the ignored prevail conditions to some degree. The work by Steinmetz and Hoffmann (2016) on clause learning is also related. They compute the critical path heuristic h^C for a collection of atom conjunctions C (Keyder, Hoffmann, and Haslum, 2014) and add conjunctions to C during the search. In a potential heuristic, such conjunctions could be added as new features.

19. Summary

We introduced potential heuristics as an interesting new class of heuristics. They fix the mathematical structure of the heuristic function to a simple weighted sum over state features. Every heuristic function can be expressed in this way if the features are complex enough, but even simple atomic features are sufficient to express useful heuristic functions.

Having a fixed mathematical structure for the whole class of heuristics makes it possible to express desired properties of the heuristic in a declarative way. For example, the conditions under which a potential heuristic is admissible and consistent can be expressed with linear constraints over the feature weights. The constraints are necessary and sufficient, so they *characterize* the subset of all potential heuristics that are admissible and consistent. This is different from most other heuristics, which are usually derived from some insight into the planning tasks and then shown to have properties like admissibility (or not). In contrast, with potential heuristics we declaratively state the properties we want the heuristic to have and then use an LP solver to find such a heuristic. Every weight function that satisfies the constraints has the desired property (in this case admissibility and consistency). With a definition of what makes a heuristic function "good" we can select *the best* heuristic from this set. We only considered potential heuristics for optimal search here. For inadmissible heuristics, higher heuristic values are not necessarily better and even the heuristic error should not be used to define the quality of a heuristic. For example, multiplying all heuristic values with a large constant greatly changes the heuristic error but has no effect on the order of node expansions in a greedy best-first search (GBFS). Wilt and Ruml (2015) investigate quality measures of heuristics for GBFS based on rank correlation. Using such quality measures to synthesize potential heuristics is an interesting topic for future research.

There are compact characterizations of admissible and consistent potential heuristics as a set of linear constraints if features are atomic or binary. For larger features this is no longer true. Testing whether a general potential function is consistent is already co-NP-hard, even if it only uses ternary features. This implies that there can be no compact characterization for ternary or larger features in general unless P $=$ NP. However, we also presented a fixed-parameter tractable method to find admissible and consistent potential heuristics that is only exponential in a parameter that measures how interdependent the features are.

Admissible and consistent atomic potential functions can be seen as a dual view on the state equation heuristic. The main difference is that h^{SEQ} is optimized in every state while potential heuristics are optimized only once to find weights. Re-optimizing an

atomic potential heuristic in every state would make it identical to h^{SEQ}. This becomes clearer in the context of cost partitioning. A potential heuristic that uses abstract states of a set of abstractions as its features computes a general transition cost partitioning over the flow heuristics on the abstractions. If the potential heuristic maximizes the heuristic value of a state, then the cost partitioning is optimal for this state. Since the weights are not changed during the search, potential heuristics can be seen as computing a cost partitioning that is optimal for a certain situation (the initial state, sampled states, etc.) and then using this cost partitioning throughout the search.

Since potential functions are so quick to evaluate, using several heuristics that are optimized for different parts of the search space works very well. The maximum of several potential heuristics combines their strengths without a large impact on the evaluation time. From the perspective of cost partitioning, this corresponds to computing multiple cost partitionings, optimizing them for different parts of the search space and using all of them during the search.

Potential functions can also be used for other purposes besides heuristic functions. They naturally group heuristic functions into equivalence classes according to the size of features required to represent them. We used this to define the *correlation complexity* of planning domains, which measures how complex a heuristic has to be so a hill-climbing search can solve tasks without backtracking.

Finally, we used potential functions to detect unsolvability. Similar to admissible potential heuristics, the desired property of the heuristic can be specified declaratively and a solver can be used to test if a function with this property exists. In this case, the desired property is that the function separates the search space into two regions, one including all reachable states and one including the goal state. If such a function exists, the task is unsolvable.

The potential heuristics we evaluated here only used two sets of features: all atomic features or all binary and atomic features. We found that a maximum over three atomic potential heuristics already quite closely approximates h^{SEQ}, which is an upper bound on any atomic potential heuristic. Using more atomic potential functions, this gap can be closed even more (Seipp, Pommerening, and Helmert, 2015). Beyond that, there is little room for improvement for atomic potential functions and larger features are required to achieve higher heuristic values. With the full set of binary and atomic features, heuristic values are higher but the optimization is often prohibitively expensive. Finding a good middle ground here is an interesting area for future research. There are many ways good feature sets could be discovered. For example, features could be derived from the task's causal graph, learned from smaller tasks, or detected during the search similar to clause learning in SAT.

Part IV.

Conclusion

20. Conclusion

We started with the question how admissible heuristic estimates can be combined. Previously, maximization, the canonical heuristic, and non-negative operator cost partitioning were the well-known answers to this question. In Part I we offered two new answers by extending cost partitioning to general cost functions and to transition cost functions. The extensions are orthogonal and both make the heuristic combination more powerful. For example, general cost partitioning can be used to show that a task is unsolvable, even in cases where all component heuristics have a finite value under the original cost function. It also benefits from heuristics that have a heuristic value of 0 under the original cost function (such as projections to non-goal variables) and can derive exponential heuristic values from polynomially sized abstractions. All of this is not possible with non-negative cost partitioning. An optimal general operator cost partitioning of explicit state abstraction heuristics can be computed in polynomial time. The effort is comparable to computing an optimal non-negative operator cost partitioning, which uses more constraints but has less complex interactions. Finding an optimal transition cost partitioning on the other hand, takes exponential time in the task description size in general but can also increase the resulting heuristic value.

Operator counting was introduced in Part II and offers a new perspective on general operator cost partitioning. An operator-counting LP heuristic for a set of operator-counting constraints computes an optimal general operator cost partitioning over the LP heuristics for individual operator-counting constraints. The corresponding MIP heuristic can achieve even higher values. This means that all heuristics that can be expressed in this framework can be combined with each other. We have seen that the cost partitioning between different types of constraints often gives better estimates than the maximization between them. For example, combining the constraints for the state equation heuristic and LM-cut landmarks can be interpreted as a cost partitioning over atomic flow heuristics and the landmarks. This combination is a state-of-the-art heuristic and solves more tasks than either LM-cut or the state equation heuristic alone.

In addition to giving a new view on cost partitioning, the operator-counting framework is interesting as a way of reasoning about heuristics. We showed that many existing heuristics can be expressed as operator-counting heuristics, which enabled us to analyze and compare the generated LP constraints. For example, we showed that the safety-based improvement cannot increase the basic state equation heuristic estimate. We also analyzed the constraints of flow heuristics and showed how they can be simplified. This relates flow heuristics like the state equation heuristic and its extensions to cost-partitioned abstraction heuristics and explains their superior performance.

Part III introduced potential heuristics, which offer a new perspective on general transition cost partitioning. Admissible and consistent potential heuristics can be thought of as computing such a cost partitioning once and then re-using it for every state. This makes them extremely fast to evaluate during the search but less accurate in areas of the search space they are not optimized for. Using more than one potential heuristic mitigates this effect by combining the strengths of all of them. We introduced compact sets of linear constraints that characterize admissible and consistent potential heuristics for atomic and binary features. Having such a characterization means that it is possible to select the best admissible and consistent potential function for a given definition of "best". Solving a single LP is sufficient to select the best heuristic if the quality measure can be defined as a linear expression over feature weights, such as maximizing the initial heuristic value, or the average heuristic value of all or some sampled states. We further showed that such compact characterizations do not exist for larger features in general (even when just using ternary features) but they can be found with a fixed-parameter tractable method if features are not too interdependent.

Both operator-counting and potential heuristics are declarative in a way. In contrast to heuristics that require the deep insight of an expert to develop, their heuristic estimates are derived by specifying simple properties of the problem and delegating the hard task of combining this knowledge to an LP or MIP solver.

In case of operator-counting heuristics, the declared properties are necessary properties of plans expressed in terms of linear constraints over operator-counting variables. We have already seen extensions of the framework where other properties are considered. For example, Imai and Fukunaga (2014) used necessary properties of delete-relaxed plans, and Trevizan, Thiébaux, and Haslum (2017) used necessary properties of optimal strategies for probabilistic planning tasks. Extensions like these are conceivable for many other planning formalisms and could also be used to connect them to each other. Such heuristics could reason about classical, numerical, temporal and probabilistic aspects of a planning task at the same time.

In case of potential heuristics, the declared properties are requirements on the form of the heuristic function, such as admissibility and consistency. We also discussed requirements for separating functions instead, which use invariants to prove that a task is unsolvable.

Potential and operator-counting heuristics offer many interesting question for future research. This is particularly true when considering abstractions, where operator counting with flow constraints corresponds to general operator cost partitioning, and admissible and consistent potential heuristics correspond to general transition cost partitionings. Finding a good collection of abstractions for those heuristics is a challenging problem. There has been much research on finding good sets of abstractions (e.g. Haslum, Bonet, and Geffner, 2005; Edelkamp, 2006; Haslum et al., 2007; Pommerening, Röger, and Helmert, 2013; Fan, Müller, and Holte, 2014; Seipp and Helmert, 2014; Sievers, Wehrle, and Helmert, 2016). However, how useful a set of abstractions is also depends on the way heuristic estimates are combined (for an example, see Table 6.3). The set of

all atomic projections already provided good heuristics estimates for both potential and operator-counting heuristics, but we also saw that extending this set (e.g. with partial merges of h^{SEQ}) can be very beneficial. Finding abstraction collections that specifically work well in the context of general operator of transition cost partitioning is thus an interesting topic.

As we have often seen a trade-off between heuristic accuracy and computation time, dynamically finding a good balance also is an interesting research question. Operator-counting constraints are well suited for this because an operator-counting heuristic always returns admissible estimates, while every added constraint makes the heuristic potentially more accurate but also more expensive to compute. For potential heuristics, a similar trade-off can be made be either adding more features to the heuristics and re-optimizing their weights or by computing a new potential function and adding it to a set of heuristic functions. This could also be done during the search, for example by detecting and analyzing heuristic errors to identify promising new features for areas of the search space where heuristic quality can be improved.

Appendix A.

Proof of Theorem 10.1

In Section 10.1 we showed that operator-counting LP heuristics compute an optimal general operator cost partitioning. Specifically, if $\mathcal{C}$ is a constraint set for a state s, then

$$h^{\text{LP}}_{\mathcal{C}}(\mathit{cost}) = h^{\text{OCP}}_{\{h^{\text{LP}}_{\{C\}} | C \in \mathcal{C}\}}(s).$$

We ignored the fact that operator-counting constraints may contain auxiliary variables in the proof sketch of Theorem 10.1 and glossed over some details. We now repeat the proof in more detail to demonstrate that these are unproblematic.

In the following, we assume that there are n non-negative operator-counting variables $\mathbf{Count} = (\mathsf{Count}_{o_1}, \ldots, \mathsf{Count}_{o_n})^\top$. We further assume that each constraint $C \in \mathcal{C}$ consists of k_C inequalities over $\mathbf{Count}$ and m_C non-negative auxiliary variables $\mathbf{Aux}^C = (\mathsf{Aux}^C_1, \ldots, \mathsf{Aux}^C_{m_C})^\top$. If the constraint requires unrestricted auxiliary variables, they can be replaced by two non-negative variables as shown in Chapter 3. We write *count-coeffs*^{C} for the $(n \times k_C)$ matrix of coefficients that determine the influence on the operator-counting variables, *aux-coeffs*^{C} for the $(m_C \times k_C)$ matrix of coefficients that determine the influence on the auxiliary variables, and *bounds*^{C} for the k_C-vector of bounds. Then C is the constraint

$$\mathit{count\text{-}coeffs}^C \mathbf{Count} + \mathit{aux\text{-}coeffs}^C \mathbf{Aux}^C \geq \mathit{bounds}^C.$$

We now consider a *constraint set* $\mathcal{C}$ of operator-counting constraints for a state s. The linear program $\text{LP}_{\mathcal{C}}(\mathit{cost})$ computed by $h^{\text{LP}}_{\mathcal{C}}$ is:

$$\begin{aligned}
&\text{Minimize } \sum_{o \in \mathcal{O}} \mathit{cost}(o)\mathsf{Count}_o \text{ subject to} \\
\mathit{count\text{-}coeffs}^C \mathbf{Count} + \mathit{aux\text{-}coeffs}^C \mathbf{Aux}^C &\geq \mathit{bounds}^C && \text{for all } C \in \mathcal{C} \\
\mathbf{Aux}^C &\geq \mathbf{0} && \text{for all } C \in \mathcal{C} \\
\mathbf{Count} &\geq \mathbf{0}
\end{aligned}$$

In this linear program, we introduce new non-negative variables LCount^C_o and new equations $\mathsf{LCount}^C_o = \mathsf{Count}_o$ for every $C \in \mathcal{C}$ and $o \in \mathcal{O}$. Replacing every occurrence of Count_o in the remaining inequalities by LCount^C_o does not influence the optimal objective value:

$$\begin{aligned}
&\text{Minimize } \sum_{o\in\mathcal{O}} \mathit{cost}(o)\mathsf{Count}_o \text{ subject to} \\
\mathit{count\text{-}coeffs}^C\mathbf{LCount}^C + \mathit{aux\text{-}coeffs}^C\mathbf{Aux}^C &\geq \mathit{bounds}^C && \text{for all } C \in \mathcal{C} \\
\mathbf{Count} - \mathbf{LCount}^C &= \mathbf{0} && \text{for all } C \in \mathcal{C} \\
\mathbf{Aux}^C \geq \mathbf{0}, \mathbf{LCount}^C &\geq \mathbf{0} && \text{for all } C \in \mathcal{C} \\
\mathbf{Count} &\geq \mathbf{0}
\end{aligned}$$

For each constraint C, the dual of this LP contains one non-negative variable Dual_i^C for each of its inequalities $1 \leq i \leq k_C$ and one unbounded variable Cost_o^C for each of its equations. The bounds vector and the objective function change their role, so objective coefficients are bounds^C for $\mathbf{Dual}^C$ and $\mathbf{0}$ for $\mathbf{Cost}^C$. The primal variables $\mathbf{Aux}^C$ only occur in the inequalities for constraint C and correspond to the dual constraints

$$\mathit{aux\text{-}coeffs}^{C\top}\mathbf{Dual}^C \leq \mathbf{0}.$$

The primal variables $\mathbf{LCount}^C$ only occur in the inequalities and equations for constraint C and correspond to the dual constraints

$$\mathit{count\text{-}coeffs}^{C\top}\mathbf{Dual}^C - \mathbf{Cost}^C \leq \mathbf{0}.$$

Each primal variable Count_o occurs in exactly one equation for each constraint C and corresponds to the dual constraints

$$\sum_{C\in\mathcal{C}} \mathsf{Cost}_o^C \leq \mathit{cost}(o).$$

Together, these constraints make up the dual of $\mathrm{LP}_{\mathcal{C}}(\mathit{cost})$:

$$\begin{aligned}
&\text{Maximize } \sum_{C\in\mathcal{C}} \mathit{bounds}^{C\top}\mathbf{Dual}^C \text{ subject to} \\
\mathit{count\text{-}coeffs}^{C\top}\mathbf{Dual}^C &\leq \mathbf{Cost}^C && \text{for all } C \in \mathcal{C} && \text{(A.1)} \\
\mathit{aux\text{-}coeffs}^{C\top}\mathbf{Dual}^C &\leq \mathbf{0} && \text{for all } C \in \mathcal{C} && \text{(A.2)} \\
\sum_{C\in\mathcal{C}} \mathsf{Cost}_o^C &\leq \mathit{cost}(o) && \text{for all } o \in \mathcal{O} && \text{(A.3)} \\
\mathbf{Dual}^C &\geq \mathbf{0} && \text{for all } C \in \mathcal{C} && \text{(A.4)}
\end{aligned}$$

This is a cost partitioning because the components are independent of each other given fixed values for the variables $\mathbf{Cost}$. To show this formally, we introduce some abbreviating notation. For a constraint $C \in \mathcal{C}$, we define $\mathit{obj}^C(\mathbf{Dual}^C) = \mathit{bounds}^C\mathbf{Dual}^C$ and $\mathit{constr}^C(\mathbf{Dual}^C, \mathbf{Cost}^C)$ as the set of all constraints (A.1) and (A.2) for C. We also

define $CP(\mathbf{Cost}^C)$ as the set of all constraints (A.3), i.e. the cost partitioning property. The problem then can be rewritten as follows.

$$\max_{\substack{\mathbf{Cost}^C \in \mathbb{R} \\ \mathbf{Dual}^C \in \mathbb{R}^+}} \left\{ \sum_{C \in \mathcal{C}} \mathit{obj}^C(\mathbf{Dual}^C) \;\middle|\; \mathit{constr}^C(\mathbf{Dual}^C, \mathbf{Cost}^C) \text{ for } C \in \mathcal{C} \text{ and } CP(\mathbf{Cost}^C) \right\}$$

$$= \max_{\mathbf{Cost}^C \in \mathbb{R}} \left\{ \sum_{C \in \mathcal{C}} \max_{\mathbf{Dual}^C \in \mathbb{R}^+} \left\{ \mathit{obj}^C(\mathbf{Dual}^C) \mid \mathit{constr}^C(\mathbf{Dual}^C, \mathbf{Cost}^C) \right\} \;\middle|\; CP(\mathbf{Cost}^C) \right\}$$

$$= \max_{\mathbf{Cost}^C \in \mathbb{R}} \left\{ \sum_{C \in \mathcal{C}} h^C(\mathbf{Cost}^C) \;\middle|\; CP(\mathbf{Cost}^C) \right\}$$

where $h^C(\mathit{cost}^C)$ is defined as the optimal objective value of the following LP.

$$\begin{aligned} &\text{Maximize } \mathit{bounds}^C \mathbf{Dual}^C \text{ subject to} \\ &\mathit{count\text{-}coeffs}^{C\top} \mathbf{Dual}^C \leq \mathit{cost}^C \\ &\mathit{aux\text{-}coeffs}^{C\top} \mathbf{Dual}^C \leq \mathbf{0} \\ &\mathbf{Dual}^C \geq \mathbf{0} \end{aligned}$$

The dual of the LP defining $h^C(\mathit{cost}^C)$ is exactly the model of the operator-counting heuristic for C under cost function cost^C, i.e. $h^{\text{LP}}_{\{C\}}(\mathit{cost}^C)$.

$$\begin{aligned} &\text{Minimize } \sum_{o \in \mathcal{O}} \mathit{cost}^C(o) \mathsf{Count}_o \text{ subject to} \\ &\mathit{count\text{-}coeffs}^C \mathbf{Count} + \mathit{aux\text{-}coeffs}^C \mathbf{Aux}^C \geq \mathit{bounds}^C \\ &\mathbf{Count} \geq \mathbf{0} \text{ and } \mathbf{Aux}^C \geq \mathbf{0} \end{aligned}$$

We conclude that $h^C = h^{\text{LP}}_{\{C\}}$. Therefore, the model of $h^{\text{LP}}_{\mathcal{C}}$ is equivalent to

$$\max_{\mathbf{Cost}^C \in \mathbb{R}} \left\{ \sum_{C \in \mathcal{C}} h^{\text{LP}}_{\{C\}}(\mathbf{Cost}^C) \;\middle|\; CP(\mathbf{Cost}^C) \right\}.$$

The constraint *CP* is the cost partitioning property, so we can conclude

$$h^{\text{LP}}_{\mathcal{C}}(\mathit{cost}) = h^{\text{OCP}}_{\{h^{\text{LP}}_{\{C\}} \mid C \in \mathcal{C}\}}(s).$$

Appendix B.

From TNF to Unrestricted SAS$^+$

We sometimes required tasks to be in transition normal form to simplify presentation. We now show how ideas using this restriction can be generalized to unrestricted SAS$^+$.

B.1. Net Change Constraints

Without knowing the state in which an operator o is applied, it is in general not possible to tell if it produces (or consumes) a given atom. An operator with precondition $\langle V, v\rangle$ and effect $\langle V, v'\rangle$ (with $v \neq v'$) always consumes $\langle V, v\rangle$ and always produces $\langle V, v'\rangle$. However, if an operator only has the effect on V but not the precondition, it *may* produce $\langle V, v'\rangle$, but only if $s[V] \neq v'$ in the state s in which the operator is applied. Similarly, it *may* consume the current value of V, but we cannot know what this value is from the operator description alone. Similar vagaries arise from variables whose value is unspecified in the goal. However, we can give upper and lower bounds on the induced net change for arbitrary plans depending on their operator counts. To do so, we distinguish four disjoint classes of operators for each atom:

- Operators that *always produce* atom $\langle V, v\rangle$:

  $$\mathit{AP}_{\langle V,v\rangle} = \{o \in \mathcal{O} \mid \mathit{eff}(o)[V] = v \text{ and } \mathit{pre}(o)[V] = v' \text{ with } v' \neq v\}$$

- Operators that *sometimes produce* atom $\langle V, v\rangle$:

  $$\mathit{SP}_{\langle V,v\rangle} = \{o \in \mathcal{O} \mid \mathit{eff}(o)[V] = v \text{ and } V \notin \mathit{vars}(\mathit{pre}(o))\}$$

- Operators that *always consume* atom $\langle V, v\rangle$:

  $$\mathit{AC}_{\langle V,v\rangle} = \{o \in \mathcal{O} \mid \mathit{eff}(o)[V] = v' \text{ with } v \neq v' \text{ and } \mathit{pre}(o)[V] = v\}$$

- Operators that *sometimes consume* atom $\langle V, v\rangle$:

  $$\mathit{SC}_{\langle V,v\rangle} = \{o \in \mathcal{O} \mid \mathit{eff}(o)[V] = v' \text{ with } v \neq v' \text{ and } V \notin \mathit{vars}(\mathit{pre}(o))\}$$

Operators that fall into none of these classes never change the truth value of the atom.

We can extend the definitions of the net change induced by an operator or operator sequence for an atom $\langle V, v\rangle$. The definitions now depend on the state as well.

Definition B.1 (induced net change)**.** *Let o be an operator and $\pi = \langle o_1, \ldots, o_n \rangle$ an operator sequence such that o and π are applicable in a state s. The* net change that o induces for atom $\langle V, v \rangle$ *in s is*

$$\mathit{netchange}(o)^s_{\langle V,v \rangle} = \begin{cases} 1 & \text{if } o \text{ applied in } s \text{ produces } \langle V, v \rangle \\ -1 & \text{if } o \text{ applied in } s \text{ consumes } \langle V, v \rangle \\ 0 & \text{otherwise.} \end{cases}$$

The accumulated net change *induced by sequence π is*

$$\mathit{netchange}(\pi)^s_{\langle V,v \rangle} = \sum_{i=1}^{n} \mathit{netchange}(o_i)^{s[\![\langle o_1,\ldots,o_{i-1} \rangle]\!]}_{\langle V,v \rangle} .$$

With the classification of operators into $\mathit{AP}_{\langle V,v \rangle}$, $\mathit{SP}_{\langle V,v \rangle}$, $\mathit{AC}_{\langle V,v \rangle}$ and $\mathit{SC}_{\langle V,v \rangle}$, we can give bounds to the possible net change induced by an operator (sequence).

$$\mathit{netchange}(o)^s_{\langle V,v \rangle} \in \begin{cases} \{1\} & \text{if } o \in \mathit{AP}_{\langle V,v \rangle} \\ \{0,1\} & \text{if } o \in \mathit{SP}_{\langle V,v \rangle} \\ \{-1\} & \text{if } o \in \mathit{AC}_{\langle V,v \rangle} \\ \{-1,0\} & \text{if } o \in \mathit{SC}_{\langle V,v \rangle} \\ \{0\} & \text{otherwise} \end{cases}$$

This justifies the following proposition:

Proposition B.1. *The accumulated net change induced by the application of operator sequence π in s can be bounded from above and below as follows:*

$$\sum_{o \in \mathit{AP}_{\langle V,v \rangle}} \mathit{occur}_\pi(o) \; + \sum_{o \in \mathit{SP}_{\langle V,v \rangle}} \mathit{occur}_\pi(o) \; - \sum_{o \in \mathit{AC}_{\langle V,v \rangle}} \mathit{occur}_\pi(o) \; \geq \; \mathit{netchange}(\pi)^s_{\langle V,v \rangle}$$

$$\geq \sum_{o \in \mathit{AP}_{\langle V,v \rangle}} \mathit{occur}_\pi(o) \; - \sum_{o \in \mathit{AC}_{\langle V,v \rangle}} \mathit{occur}_\pi(o) \; - \sum_{o \in \mathit{SC}_{\langle V,v \rangle}} \mathit{occur}_\pi(o) \, .$$

Likewise, we can give bounds on the total net change between a given state s and any goal state: the set of *possible net change* values between such states is

$$\mathit{pnc}^{s \to \star}_{\langle V,v \rangle} = \begin{cases} \{0,1\} & \text{if } V \notin \mathit{vars}(s_\star) \text{ and } s[V] \neq v \\ \{-1,0\} & \text{if } V \notin \mathit{vars}(s_\star) \text{ and } s[V] = v \\ \{1\} & \text{if } s_\star[V] = v \text{ and } s[V] \neq v \\ \{-1\} & \text{if } s_\star[V] = v' \text{ and } s[V] = v \neq v' \\ \{0\} & \text{otherwise.} \end{cases}$$

Combined with the bounds from Proposition B.1, we can finally define a general version of lower and upper bound net change constraints:

Definition B.2 (net change constraint)**.** *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_{\mathrm{I}}, s_\star, cost \rangle$ be a planning task and s one of its states. For an atom $\langle V, v \rangle$ over $\mathcal{V}$ the* lower bound net change constraint *$c^{\mathrm{ncl}}_{s,\langle V,v\rangle}$ for atom $\langle V, v \rangle$ and state s is the constraint*

$$\sum_{o \in AP_{\langle V,v\rangle}} \mathsf{Count}_o \; + \sum_{o \in SP_{\langle V,v\rangle}} \mathsf{Count}_o \; - \sum_{o \in AC_{\langle V,v\rangle}} \mathsf{Count}_o \geq \min(pnc^{s \to \star}_{\langle V,v\rangle})$$

and the upper bound net change constraint *$c^{\mathrm{ncu}}_{s,\langle V,v\rangle}$ is the constraint*

$$\max(pnc^{s \to \star}_{\langle V,v\rangle}) \geq \sum_{o \in AP_{\langle V,v\rangle}} \mathsf{Count}_o \; - \sum_{o \in AC_{\langle V,v\rangle}} \mathsf{Count}_o \; - \sum_{o \in SC_{\langle V,v\rangle}} \mathsf{Count}_o$$

Close inspection of these constraints again shows that the set of lower bound net change constraints for all atoms exactly matches the constraints of h^{SEQ} for unrestricted SAS$^+$ tasks.

The definition for TNF tasks is much simpler, and every task can be transformed into TNF by transition normalization. Is any information lost during the transformation? It turns out that the state equation heuristic value for the unrestricted SAS$^+$ task Π is identical to the heuristic estimate in *TNF*(Π) for which we can use the simpler and cleaner constraint system.

Proposition B.2. *Let Π be a general SAS$^+$ task. Let $\mathcal{C}_\Pi$ be the set of lower bound net change constraints for all atoms of Π and a state s of Π according to Definition B.2. Let $\mathcal{C}_{TNF(\Pi)}$ be the same set of constraints for TNF(Π) according to Definition 9.8. Then,*

$$h^{\mathrm{LP}}_{\mathcal{C}_\Pi}(s) = h^{\mathrm{LP}}_{\mathcal{C}_{TNF(\Pi)}}.$$

Proof: Consider the lower bound net change constraint for an atom $\langle V, v \rangle$. Operators that always produce or consume $\langle V, v \rangle$ in Π have a precondition and an effect on V which is not changed by transition normalization. In *TNF*(Π) they induce the same coefficients in the constraint for $\langle V, v \rangle$.

Operators that sometimes produce $\langle V, v \rangle$ in Π have an additional precondition $\langle V, \mathbf{u} \rangle$ and thus always produce $\langle V, v \rangle$ in *TNF*(Π). Since operators that always produce a atom are treated the same as operators that sometimes produce it, this does not affect the equation.

Operators that sometimes consume $\langle V, v \rangle$ in Π also have an additional precondition $\langle V, \mathbf{u} \rangle$ in *TNF*(Π). This precondition guarantees that they cannot consume $\langle V, v \rangle$ in *TNF*(Π) so they are irrelevant for this atom. In both Π and *TNF*(Π) they do not occur in the constraint.

Operators that have a prevail condition on $\langle V, v \rangle$ have the matching effect in *TNF*(Π). They can be either left out of the constraint completely or they can be seen as both consumers and producers, in which case their terms cancel out.

Operators that are irrelevant in Π because they do not mention V are still irrelevant in *TNF*(Π).

Finally, the operators $\mathit{forget}_{\langle V,v\rangle}$ in $\mathit{TNF}(\Pi)$ occur in two constraints: they always consume $\langle V, v\rangle$ and always produce $\langle V, \mathbf{u}\rangle$. As a result, the linear programs solved for Π and $\mathit{TNF}(\Pi)$ differ in only two aspects. Every constraint for $\langle V, v\rangle$ is extended by $-\,\mathsf{Count}_{\mathit{forget}_{\langle V,v\rangle}}$ and one additional constraint for $\langle V, \mathbf{u}\rangle$ is added:

$$\sum_{v\in \mathit{dom}(V)} \mathsf{Count}_{\mathit{forget}_{\langle V,v\rangle}} \geq [s_\star[V] = \mathbf{u}].$$

This constraint is redundant for goal variables. For non-goal variables it states that the slack from all inequalities must sum up to at least one. As the constraints describe a network flow, starting with an activation of one at the initial value of V, this is always satisfied. Without redundant constraints, each variable $\mathsf{Count}_{\mathit{forget}_{\langle V,v\rangle}}$ occurs only once and setting them to non-zero values can only make the constraints harder to satisfy. □

Analogously to Theorem 10.3, we can also show that all upper bound net change constraints for atoms are redundant in the presence of lower bound net change constraints for all atoms. The proof is analogous to the one for TNF: summing up the lower bound constraints for all atoms of a variable V with values in $\mathit{dom}(V)\setminus\{v\}$ results in the upper bound constraint for $\langle V, v\rangle$.

B.2. Flow Constraints

In Sections 9.6 and 10.4 we analyzed flow heuristics based on abstract transition systems with a unique goal state. If an SAS$^+$ planning task does not mention every variable in its goal, an abstract transition system can have more than one goal state. It is easy to generalize the minimum-cost flow problem to handle more than one sink: we add an artificial sink and 0-cost edges from every original sink to the artificial one. In a planning task, this corresponds to generating a unique goal state and operators to reach it from the original goal states. The resulting flow model then changes as follows, where $\mathrm{TS} = \langle \mathcal{S}, \mathcal{T}, s_\mathrm{I}, \mathrm{S}_G\rangle$ is the original transition system and G_s is a new transition-counting variable for the transition from original goal state s to the artificial goal:

$$\text{Minimize} \sum_{o\in\mathcal{O}} \sum_{t\in \mathit{trans}_\mathrm{TS}(o)} \mathit{cost}(o)\mathsf{Count}_t \text{ subject to}$$

$$\sum_{t\in \mathit{in}_\mathrm{TS}(s)} \mathsf{Count}_t - \sum_{t\in \mathit{out}_\mathrm{TS}(s)} \mathsf{Count}_t = -[s = s_\mathrm{I}] \qquad \text{for all } s \in \mathcal{S}\setminus \mathrm{S}_G \tag{B.1}$$

$$\sum_{t\in \mathit{in}_\mathrm{TS}(s)} \mathsf{Count}_t - \sum_{t\in \mathit{out}_\mathrm{TS}(s)} \mathsf{Count}_t - \mathsf{G}_s = -[s = s_\mathrm{I}] \qquad \text{for all } s \in \mathrm{S}_G \tag{B.2}$$

$$\sum_{s\in \mathrm{S}_G} \mathsf{G}_s = 1 \tag{B.3}$$

$$\mathsf{Count}_t \geq 0 \qquad \text{for all } t \in \mathcal{T} \tag{B.4}$$

$$\mathsf{G}_s \geq 0 \qquad \text{for all } s \in \mathrm{S}_G \tag{B.5}$$

All simplification rules refer only to the structure of the abstract transition system and therefore are also applicable to unrestricted SAS$^+$ tasks. In particular, Rule 6 can be used to remove the artificial goal state. Rule 6 may still only be used once, but we can consider abstractions that map all states we want to remove to the same abstract state first, and then remove this abstract state. Once constraint (B.3) is removed from the above LP, the artificial transitions G_s can be removed as well, by turning equations (B.2) into inequalities:

$$\text{Minimize} \sum_{o \in \mathcal{O}} \sum_{t \in \mathit{trans}_{\mathrm{TS}}(o)} \mathit{cost}(o)\mathsf{Count}_t \text{ subject to}$$

$$\sum_{t \in \mathit{in}_{\mathrm{TS}}(s)} \mathsf{Count}_t - \sum_{t \in \mathit{out}_{\mathrm{TS}}(s)} \mathsf{Count}_t = -[s = s_{\mathrm{I}}] \qquad \text{for all } s \in \mathcal{S} \setminus \mathrm{S}_G \qquad \text{(B.6)}$$

$$\sum_{t \in \mathit{in}_{\mathrm{TS}}(s)} \mathsf{Count}_t - \sum_{t \in \mathit{out}_{\mathrm{TS}}(s)} \mathsf{Count}_t \geq -[s = s_{\mathrm{I}}] \qquad \text{for all } s \in \mathrm{S}_G \qquad \text{(B.7)}$$

$$\mathsf{Count}_t \geq 0 \qquad \text{for all } t \in \mathcal{T} \qquad \text{(B.8)}$$

Proposition 10.5 points out how the constraints of the state equation heuristic can be seen as simplified flow constraints for atomic projections. The mentioned simplification rules 2 and 3 are not sufficient to achieve the same model as the state equation heuristic for unrestricted SAS$^+$ tasks. We know from Proposition B.2 that the heuristic values of the two models must be the same, but the simplified constraints introduce more LP constraints and LP variables. Operators that have an effect but no precondition on a variable V introduce several state-changing transitions and one self-loop in the projection on V. While the flow constraints have transition-counting variables and linking constraints for such operators, the state equation heuristic treats them as sometimes producing their effect and sometimes consuming every other value of V. The lower bound net change constraints then only mention the operator-counting variable in the constraint for the effect and contain no transition-counting variables or linking constraints. This structure could be used to derive another simplification rule.

Handling operators that sometimes produce or consume values in larger projections is even more complicated, e.g. when considering (partial) merges of values in Propositions 10.6, 10.7 and 10.8. In larger projections, non-TNF operators induce a mixture of self-loops and state changing transitions that do not necessarily all end in one abstract state. Investigating such transitions could lead to interesting further simplification rules.

B.3. Atomic Potential Heuristics

In Section 15.1 we derived a set of constraints characterizing admissible and consistent potential functions for atomic features in TNF tasks. We now show that such constraints can be derived for unrestricted SAS$^+$ tasks and that they match the constraints derived from the transition normalization of the task. This means that the simpler definition

for tasks in TNF is sufficient and can be used for general tasks by means of transition normalization.

For a partial variable assignment p, we introduce the notation $\mathit{wmax}(V, p)$ for the maximal weight that an atom of variable V can contribute to the value of a state consistent with p:

$$\begin{aligned}\mathit{wmax}(V, p) &= \max_{s \models p} w(\langle V, s[V] \rangle) \\ &= \begin{cases} w(\langle V, p[V] \rangle) & \text{if } V \in \mathit{vars}(p) \\ \max_{v \in \mathit{dom}(V)} w(\langle V, v \rangle) & \text{otherwise} \end{cases}\end{aligned}$$

A potential heuristic h^w with weights w is goal-aware if and only if $h^w(s_G) \leq 0$ for all states s_G consistent with $s_\star$. It is sufficient to require this condition only for a state that is consistent with $s_\star$ and has maximal potential among all goal states:

$$\max_{s_G \models s_\star} h^w(s_G) = \sum_{V \in \mathcal{V}} \mathit{wmax}(V, s_\star) \leq 0$$

The resulting inequality is not state-dependent and is necessary and sufficient for h^w to be goal-aware.

A potential heuristic h^w is consistent if and only if $h^w(s) \leq \mathit{cost}(o) + h^w(s[\![o]\!])$ for every state s and every operator o applicable in s. This condition can be simplified as follows because all weights of atoms that are not changed by an effect cancel out:

$$\begin{aligned}\mathit{cost}(o) &\geq \sum_{V \in \mathcal{V}} w(\langle V, s[V] \rangle) - \sum_{V \in \mathcal{V}} w(\langle V, s[\![o]\!][V] \rangle) \\ &= \sum_{V \in \mathit{vars}(\mathit{eff}(o))} (w(\langle V, s[V] \rangle) - w(\langle V, \mathit{eff}(o)[V] \rangle)) \\ &= \sum_{\langle V, v \rangle \in \mathit{eff}(o)} (w(\langle V, s[V] \rangle) - w(\langle V, v \rangle))\end{aligned}$$

Again, it is sufficient to require the inequality only for a state that is consistent with the operator's precondition and that has maximal potential:

$$\mathit{cost}(o) \geq \sum_{\langle V, v \rangle \in \mathit{eff}(o)} (\mathit{wmax}(V, \mathit{pre}(o)) - w(\langle V, v \rangle))$$

The resulting inequality is no longer state-dependent and is necessary and sufficient for consistency. Goal-aware and consistent potential heuristics for atomic features can thus

be compactly characterized as a set of linear equations.

$$\mathsf{W}_{\langle V,v\rangle} \leq \mathsf{Max}_V \qquad \text{for all } \langle V,v\rangle \in \mathcal{A} \tag{B.9}$$

$$\sum_{V\in\mathcal{V}} \mathit{wmax}(V, s_\star) \leq 0 \tag{B.10}$$

$$\sum_{\langle V,v\rangle\in \mathit{eff}(o)} \big(\mathit{wmax}(V, \mathit{pre}(o)) - \mathsf{W}_{\langle V,v\rangle}\big) \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O} \tag{B.11}$$

$$\text{where } \mathit{wmax}(V,p) = \begin{cases} \mathsf{W}_{\langle V,p[V]\rangle} & \text{if } V \in \mathit{vars}(p) \\ \mathsf{Max}_V & \text{otherwise} \end{cases}$$

Proposition B.3. *The solutions of constraints* (B.9)–(B.11) *projected to the variables* W *are exactly the weight functions of admissible and consistent atomic potential functions.*

As a reminder, the constraints that we used to characterize admissible and consistent potential functions for TNF tasks are:

$$\sum_{\langle V,v\rangle\in s_\star} \mathsf{W}_{\langle V,v\rangle} \leq 0 \tag{B.12}$$

$$\sum_{\langle V,v\rangle\in \mathit{pre}(o)} \mathsf{W}_{\langle V,v\rangle} \;-\; \sum_{\langle V,v\rangle\in \mathit{eff}(o)} \mathsf{W}_{\langle V,v\rangle} \leq \mathit{cost}(o) \qquad \text{for all } o \in \mathcal{O} \tag{B.13}$$

Proposition B.4. *Let* Π *be a SAS*$^+$ *planning task. The constraints* (B.9)–(B.11) *for* Π *are isomorphic to the constraints* (B.12)–(B.13) *for TNF*(Π).

Proof: Transition normalization introduces a new value $\mathbf{u}$ for every variable V and an operator $\mathit{forget}_{\langle V,v\rangle}$ for every atom $\langle V,v\rangle$. The new operator leads to the constraint $\mathsf{W}_{\langle V,v\rangle} - \mathsf{W}_{\langle V,\mathbf{u}\rangle} \leq \mathit{cost}(\mathit{forget}_{\langle V,v\rangle}) = 0$, or equivalently $\mathsf{W}_{\langle V,v\rangle} \leq \mathsf{W}_{\langle V,\mathbf{u}\rangle}$. Replacing $\mathsf{W}_{\langle V,\mathbf{u}\rangle}$ with Max_V shows the theorem. □

B.4. Binary and Higher-Dimensional Potential Heuristics

With higher-dimensional potential heuristics, we grouped the set of features into three sets: irrelevant, context-independent, and context-dependent features. We can do the same for unrestricted SAS$^+$ tasks. However, in the general case, we also have to consider a feature f context-dependent for an operator o if o sometimes produces or sometimes consumes f, i.e. if o has an effect but no precondition on a variable mentioned in f. With this extended definition, irrelevant features and context-independent features behave just like in the TNF case.

Context-dependent features are harder to handle in general. For them, we have to find a compact characterization of solutions to the following equation (see Section 15.3.2).

$$Z_o \geq \max_{s \models pre(o)} \sum_{f \in \mathcal{F}_o^{\text{ctx}}} w(f)\Delta_o(f, s)$$

In Section 15.3.2 we did this by transforming the maximum into a function maximization problem over the functions

$$\psi_o^f(p) = \mathsf{W}_f([pre(o) \cup p \models f] - [\mathit{eff}(o) \cup p \models f])$$

where p is a partial variable assignment over variables mentioned by f but not by o. Since the context can now depend on variables mentioned in the effect of o but not in the precondition, we have to extend this definition.

Let $\mathcal{V}_{f \setminus o}$ be the variables mentioned by f but not by o, and let $\mathcal{V}_{f \setminus pre(o)}$ be the variables mentioned by f and o, but not in the precondition of o. Further, let $\mathit{post}(o)$ be the partial state with $\mathit{post}(o)[V] = \mathit{eff}(o)[V]$ if $V \in \mathit{vars}(\mathit{eff}(o))$ and $\mathit{post}(o)[V] = pre(o)[V]$ if $V \in \mathit{vars}(pre(o)) \setminus \mathit{vars}(\mathit{eff}(o))$, i.e. the effect of o together with all prevail conditions of o. We can then change the definition of ψ to a function from $\mathcal{V}_{f \setminus o} \cup \mathcal{V}_{f \setminus pre(o)}$ to $\{\mathsf{W}_f, -\mathsf{W}_f, 0\}$:

$$\psi'^f_o(p) = \mathsf{W}_f([pre(o) \cup p \models f] - [\mathit{post}(o) \cup p|_{\mathcal{V}_{f \setminus o}} \models f])$$

The function maximization problem over those functions can be combined with the remaining constraint as in Section 15.3.2 to get a general definition of higher-dimensional potential heuristics.

Appendix C.

Maximizing a Sum of Functions

We now leave the planning formalism and consider a technique from constraint optimization called *bucket elimination* (Dechter, 1999, 2003). We will generalize the technique and show how linear programming can be used to solve the generalization.

First, we need some basic notation. Let $\mathcal{X}$ be a set of finite-domain variables and $\mathit{dom}(X)$ the domain of each $X \in \mathcal{X}$. We extend the definition of *dom* to sets $S \subseteq \mathcal{X}$ such that $\mathit{dom}(S)$ is the set of variable assignments that map each variable in S to a value in its domain. Let $\mathbb{V}$ be a set of values. For now, think of $\mathbb{V}$ as the set of real numbers $\mathbb{R}$. We will consider more general sets later. We only require that maximization and addition is defined on values in $\mathbb{V}$, that both operations are commutative and associative, and that $\max\{a+c, b+c\} \equiv c + \max\{a, b\}$ for all $a, b, c \in \mathbb{V}$. We use the common extensions of sums and maxima from binary functions to functions of arbitrary arity and use common mathematical abbreviations like $\sum_{x\in\{1,\ldots,n\}} f(x) = f(1)+\cdots+f(n)$ and $\max_{x\in\{1,\ldots,n\}} f(x) = \max\{f(1),\ldots,f(n)\}$. As corner cases, the sum and maximum over the singleton set $\{x\}$ are x.

Let Ψ be a set of *scoped functions*, i.e. tuples $\langle S, f\rangle$ containing a *scope* $S \subseteq \mathcal{X}$ and a function $f : S \to \mathbb{V}$. For a scope S and a variable assignment $\nu \in \mathit{dom}(\mathcal{X})$ we use $\nu|_S$ to denote the projection of ν to the variables in S.

We are interested in a maximal object that the sum of all functions in Ψ can take under a common variable assignment:

$$\mathit{Max}(\Psi) = \max_{\nu\in\mathit{dom}(\mathcal{X})} \sum_{\langle S,f\rangle\in\Psi} f(\nu|_S).$$

Example Consider the set $\mathcal{X} = \{x, y\}$ of two binary variables and the scoped functions $\Psi = \{\langle\{x\}, f\rangle, \langle\{x, y\}, g\rangle\}$ of the two functions f and g with:

$f(x)$	$x=0$	$x=1$
	3	2

$g(x,y)$	$x=0$	$x=1$
$y=0$	2	-5
$y=1$	0	4

In this case, the variable assignment $\nu = \{x \mapsto 1, y \mapsto 1\}$ achieves the highest value and $\mathit{Max}(\Psi) = 2 + 4 = 6$.

Computing $\mathit{Max}(\Psi)$ is the goal of constraint optimization for extensional constraints (Dechter, 2003), an important problem in AI. It is challenging because the number of variable assignments in $\mathit{dom}(\mathcal{X})$ is exponential in the number of variables. *Bucket elimination* (Dechter, 2003) is a well-known algorithm to calculate $\mathit{Max}(\Psi)$. For reasons that will become clear later, we describe the bucket elimination algorithm in a slightly unusual way: in our formulation, the algorithm generates a system of equations, and its output can be extracted from the (uniquely defined) solution to these equations. The system of equations makes use of auxiliary variables $\mathsf{Aux}_1, \ldots, \mathsf{Aux}_m$ that take values from $\mathbb{V}$. The generated equations have the form $\mathsf{Aux}_i = \max_{j \in \{1,\ldots,k_i\}} e_{i,j}$, where $e_{i,j}$ is a sum that contains only values from $\mathbb{V}$ or the variables $\mathsf{Aux}_1, \ldots, \mathsf{Aux}_{i-1}$. Solutions to the system of equations guarantee that $\mathsf{Aux}_m \equiv \mathit{Max}(\Psi)$.

C.1. Bucket Elimination

We now describe the general algorithm and prove its correctness. Its execution depends on an order $\sigma = \langle X_1, \ldots, X_n \rangle$ of the variables in $\mathcal{X}$. The algorithm maintains a set of $|\mathcal{X}| + 1$ *buckets* $B_0, \ldots, B_n$. Bucket B_i for $1 \leq i \leq n$ contains scoped functions $\langle S, f \rangle$ such that X_i is the largest variable in S according to σ. Bucket B_0 contains functions with an empty scope. The algorithm operates in stages which are enumerated in a decreasing manner, starting at stage $n + 1$ and ending at stage 0:

- Stage $n+1$ (Initialization). Start with a set $\{B_i\}_{i=0}^{n}$ of *empty* buckets. Place each $\langle S, \psi \rangle \in \Psi$ into the bucket B_i if X_i is the largest variable in S, according to σ, or into the bucket B_0 if $S = \emptyset$. The resulting system of equations is initialized to an empty system.

- Stages $i = n, \ldots, 1$ (Elimination). Stage i eliminates variable X_i from the functions in bucket B_i.

  Let $\langle S_j, \psi_j \rangle$ for $j \in \{1, \ldots, k_i\}$ be the scoped functions currently in bucket B_i. Construct the scope $S_{X_i} = (\bigcup_{j \in \{1,\ldots,k_i\}} S_j) \setminus \{X_i\}$ that contains all variables relevant to X_i. Then construct the new function $\psi_{X_i} : S_{X_i} \to \mathbb{V}$ that represents the contribution of all functions that depend on X_i. The definition of ψ_{X_i} is added to the generated system of equations by adding one auxiliary variable $\mathsf{Aux}_{X_i,\nu}$ for every $\nu \in \mathit{dom}(S_{X_i})$ which represents the value $\psi_{X_i}(\nu)$:

  $$\mathsf{Aux}_{X_i,\nu} = \max_{x_i \in \mathit{dom}(X_i)} \sum_{j \in \{1,\ldots,k_i\}} \psi_j(\nu_{x_i}|_{S_j})$$

  where $\nu_{x_i} = \nu \cup \{X_i \mapsto x_i\}$ extends the variable assignment ν with $X_i \mapsto x_i$. If ψ_j is a function in Ψ, then $\psi_j(\nu_{x_i}|_{S_j})$ is an element of $\mathbb{V}$. Otherwise ψ_j is a previously defined function $\psi_{X_{i'}}$ for $i' > i$ and its value for $\nu' = \nu_{x_i}|_{S_j}$ is represented by $\mathsf{Aux}_{X_{i'},\nu'}$.

The newly defined function ψ_{X_i} no longer depends on X_i but depends on all variables in S_{X_i}, so $\langle S_{X_i}, \psi_{X_i} \rangle$ is added to bucket B_j if X_j is the largest variable in S_{X_i} according to σ or to B_0 if $S_{X_i} = \emptyset$. Observe that $j < i$ because S_{X_i} only contains variables from scopes where X_i is the largest variable and S_{X_i} does not contain X_i.

Finally, clear bucket B_i.

- Stage 0 (Termination). Let $\langle S_j, \psi_j \rangle$ for $j \in \{1, \dots, k\}$ be the scoped functions that currently are in bucket B_0. Add the auxiliary variable Aux_Ψ and the equation $\mathsf{Aux}_\Psi = \sum_{j \in \{1,\dots,k\}} \psi_j$, then clear bucket B_0. This is analogous to the elimination step. (All S_j are empty and the maximum is over $dom(\emptyset) = \{\emptyset\}$.)

Example Consider the functions f and g from the previous example again. Bucket elimination generates the following equations for the variable order $\sigma = \langle X, Y \rangle$.

$$\begin{aligned}
\mathsf{Aux}_1 = \mathsf{Aux}_{Y,\{X \mapsto 0\}} &= \max_{y \in \{0,1\}} g(0, y) \\
&= \max\{g(0,0), g(0,1)\} \\
\mathsf{Aux}_2 = \mathsf{Aux}_{Y,\{X \mapsto 1\}} &= \max_{y \in \{0,1\}} g(1, y) \\
&= \max\{g(1,0), g(1,1)\} \\
\mathsf{Aux}_3 = \mathsf{Aux}_{X,\emptyset} &= \max_{x \in \{0,1\}} (f(x) + \mathsf{Aux}_{Y,\{X \mapsto x\}}) \\
&= \max\{f(0) + \mathsf{Aux}_1, f(1) + \mathsf{Aux}_2\} \\
\mathsf{Aux}_4 = \mathsf{Aux}_\Psi &= \mathsf{Aux}_{X,\emptyset} = \mathsf{Aux}_3
\end{aligned}$$

Plugging in the values of f and g gives the system of equations

$$\begin{aligned}
\mathsf{Aux}_1 &= \max\{2, 0\} \\
\mathsf{Aux}_2 &= \max\{-5, 4\} \\
\mathsf{Aux}_3 &= \max\{3 + \mathsf{Aux}_1, 2 + \mathsf{Aux}_2\} \\
\mathsf{Aux}_\Psi &= \mathsf{Aux}_3
\end{aligned}$$

which has a unique solution with $\mathsf{Aux}_\Psi = 6$.

Let us convince ourselves that the output of the algorithm always has the desired form:

Proposition C.1. *Let Ψ be a set of scoped functions with codomain $\mathbb{V}$. The system of equations generated by bucket elimination on the problem Ψ has the form*

$$\mathsf{Aux}_i = \max_{j \in \{1,\dots,k_i\}} e_{i,j} \qquad \textit{for all } i \in \{1, \dots, m\},$$

where $e_{i,j}$ is a sum that contains only values from $\mathbb{V}$ or the variables $\mathsf{Aux}_1, \dots, \mathsf{Aux}_{i-1}$.

Proof: It is easy to see that all generated equations are maxima over sums (the definition in the termination phase can be seen as a maximum over all variables assignments for the empty scope, i.e. $\{\emptyset\}$).

The elements of $e_{i,j}$ are values of functions ψ_j that are in the bucket B_i when X_i is eliminated. If ψ_j was added to the bucket in the initialization stage, then it is a function from Ψ and its values are from $\mathbb{V}$. Otherwise the function was added by a previous elimination stage $i' > i$ and its definition is already part of the generated system of equations. Its value for evaluation ν is represented by $\mathsf{Aux}_{j,\nu}$ which was added to the system in stage i'. □

To show that the algorithm is correct, we first show that the function introduced when eliminating a bucket is equivalent to the functions contained in the bucket.

Proposition C.2. *Let Ψ be a set of scoped functions and $\langle S_1, f_1\rangle, \ldots, \langle S_n, f_n\rangle$ be the subset of functions whose scope includes a variable X. Let $S_X = (\bigcup_{j=1}^{n} S_j) \setminus \{X\}$ and $\psi_X(\nu) = \max_{x \in dom(X)} \sum_{j \in \{1,\ldots,n\}} f_j(\nu_x|_{S_j})$ for all variable assignments $\nu \in dom(S_X)$. Then*

$$Max(\Psi) \equiv Max((\Psi \setminus \{\langle S_1, f_1\rangle, \ldots, \langle S_n, f_n\rangle\}) \cup \{\langle S_X, \psi_X\rangle\}).$$

Proof: For the maximization over values of X all functions that do not depend on X are constant and can be moved out of the maximum. Evaluations of all other variables can be restricted to the scope S_X if only functions $f_1, \ldots, f_n$ are considered.

$$\begin{aligned}
Max(\Psi) &= \max_{\nu \in dom(\mathcal{X})} \sum_{\langle S,f\rangle \in \Psi} f(\nu|_S) \\
&= \max_{\nu' \in dom(\mathcal{X}\setminus\{X\})} \max_{x \in dom(X)} \sum_{\langle S,f\rangle \in \Psi} f(\nu'_x|_S) \\
&\equiv \max_{\nu' \in dom(\mathcal{X}\setminus\{X\})} \left(\sum_{\substack{\langle S,f\rangle \in \Psi \\ f \notin \{f_1,\ldots,f_n\}}} f(\nu'|_S) + \max_{x \in dom(X)} \sum_{j=1}^{n} f_j(\nu'_x|_{S_j}) \right) \\
&= \max_{\nu' \in dom(\mathcal{X}\setminus\{X\})} \left(\sum_{\substack{\langle S,f\rangle \in \Psi \\ f \notin \{f_1,\ldots,f_n\}}} f(\nu'|_S) + \psi_X(\nu'|_{S_X}) \right) \\
&= Max((\Psi \setminus \{\langle S_1, f_1\rangle, \ldots, \langle S_n, f_n\rangle\}) \cup \{\langle S_X, \psi_X\rangle\})
\end{aligned}$$

□

With Proposition C.2 we can show the correctness of a single elimination stage.

Proposition C.3. *Let Ψ_1 be the set of scoped functions in all buckets just before eliminating bucket B_i, where the value of $\psi_{X_j}(\nu)$ is defined as equivalent to $\mathsf{Aux}_{X_j,\nu}$ in the system of equations generated so far for $j > i$. Let Ψ_2 be the analogous set after the elimination step. Then $Max(\Psi_1) \equiv Max(\Psi_2)$.*

Proof: The functions in $\Psi_1 \setminus \Psi_2$ are exactly the functions whose scope includes X_i. Only one function is added and the generated equations together with Proposition C.1 guarantee that its definition is equivalent to

$$\psi_{X_i}(\nu) = \max_{x_i \in dom(X_i)} \sum_{\langle S,f\rangle \in \Psi_1 \setminus \Psi_2} f(\nu_{x_i}|_S).$$

Proposition C.2 then shows the equivalence. □

We can now show correctness of the bucket elimination algorithm:

Theorem C.1. *In the system of equations generated by the bucket elimination algorithm for a set of scoped functions Ψ the final variable generated (Aux_Ψ) is equivalent to $Max(\Psi)$.*

Proof: For $i \in \{n, \ldots, 1\}$ let Ψ_i be the set of scoped functions in all buckets before eliminating X_i, where functions ψ_{X_j} for $j > i$ are defined as equivalent to $\mathsf{Aux}_{X_j,\nu}$ in the system of equations so far. Let Ψ_0 be the analogous set after eliminating X_1, i.e. the set of functions left in B_0 after the elimination stage.

The initialization stage adds all elements of Ψ to the buckets, so $\Psi_n = \Psi$. An inductive proof using Proposition C.3 shows $Max(\Psi) = Max(\Psi_n) \equiv Max(\Psi_0)$. Since those functions all have the empty scope, the maximum over their sum is just their sum, which is the definition of Aux_Ψ generated in the termination stage. □

C.2. Bucket Elimination for Linear Expressions

As a generalization of the bucket elimination algorithm, consider $\mathbb{V}$ to be the following set $\mathbb{E}$ of mathematical expressions over a set of variable symbols $\mathcal{Y}$. For every $\mathsf{Y} \in \mathcal{Y}$ and $r \in \mathbb{R}$, the expressions Y, r, and $r\mathsf{Y}$ are in $\mathbb{E}$. If a and b are elements of $\mathbb{E}$, then the expressions $(a+b)$ and $\max\{a, b\}$ are elements of $\mathbb{E}$. There are no additional elements in $\mathbb{E}$. An assignment $f : \mathcal{Y} \to \mathbb{R}$ that maps variables to values can be extended to $\mathbb{E}$ in the straight-forward way. Two expressions $a, b \in \mathbb{E}$ are equivalent if $f(a) = f(b)$ for all assignments f. An expression is *linear* if it does not mention $\max$.

Clearly, maximization and addition are commutative, associative and all expressions $a, b, c \in \mathbb{E}$ satisfy $\max\{a + c, b + c\} \equiv c + \max\{a, b\}$. Bucket elimination therefore generates a system of equations $\mathsf{Aux}_i = \max_{j \in \{1,\ldots,k_i\}} e_{i,j}$, where all $e_{i,j}$ are sums over expressions and variables $\mathsf{Aux}_{i'}$ with $i' < i$. Since a variable is a mathematical expression, the whole result can be seen as a system of equations over the variables $\mathcal{Y} \cup \{\mathsf{Aux}_1, \ldots, \mathsf{Aux}_m\}$. If additionally all functions in Ψ only produce linear expressions over $\mathcal{Y}$, then in the resulting system all $e_{i,j}$ are linear expressions over $\mathcal{Y} \cup \{\mathsf{Aux}_1, \ldots, \mathsf{Aux}_m\}$.

Example Consider the example problem from the previous section again but with different definitions for f and g. We define f and g so they map to linear expressions over $\mathcal{Y} = \{\mathsf{A}, \mathsf{B}\}$:

$f(x)$	$x=0$	$x=1$
	$3\mathsf{A} - 2\mathsf{B}$	$4\mathsf{A} + 2\mathsf{B}$

$g(x,y)$	$x=0$	$x=1$
$y=0$	$8\mathsf{A}$	$-3\mathsf{B}$
$y=1$	$7\mathsf{B}$	0

In the resulting system of equations all elements in the maxima are linear expressions over the variables $\{\mathsf{A}, \mathsf{B}, \mathsf{Aux}_1, \mathsf{Aux}_2, \mathsf{Aux}_3, \mathsf{Aux}_4\}$:

$$\begin{aligned}
\mathsf{Aux}_1 &= \max\{8\mathsf{A}, 7\mathsf{B}\} \\
\mathsf{Aux}_2 &= \max\{-3\mathsf{B}, 0\} \\
\mathsf{Aux}_3 &= \max\{3\mathsf{A} - 2\mathsf{B} + \mathsf{Aux}_1, 4\mathsf{A} + 2\mathsf{B} + \mathsf{Aux}_2\} \\
\mathsf{Aux}_4 &= \mathsf{Aux}_3
\end{aligned}$$

Bucket elimination guarantees that $\mathsf{Aux}_4 \equiv \mathit{Max}(\Psi)$ for any value of A and B.

We argue that the system of equations generated by bucket elimination of a function maximization problem where all functions produce linear expressions can be solved by an LP solver, i.e. we can get rid of the (non-linear) maximization.

Theorem C.2. *Let $\mathcal{Y}$ and $\{\mathsf{Aux}_1, \ldots \mathsf{Aux}_m\}$ be disjoint sets of variables. Let $P_{\max}$ be a system of equations*

$$\mathsf{Aux}_i = \max_{j \in \{1,\ldots,k_i\}} e_{i,j} \qquad \textit{for all } i \in \{1, \ldots, m\},$$

where $e_{i,j}$ is a linear expression over variables $\mathcal{Y} \cup \{\mathsf{Aux}_1, \ldots, \mathsf{Aux}_{i-1}\}$. Let P_{LP} be the set of linear constraints

$$\mathsf{Aux}_i \geq e_{i,j} \qquad \textit{for all } i \in \{1, \ldots, m\} \textit{ and } j \in \{1, \ldots, k_i\}.$$

Every solution of $P_{\max}$ is a solution of P_{LP} and for every solution f of P_{LP} there is a solution f' of $P_{\max}$ with $f'(\mathsf{Y}) = f(\mathsf{Y})$ for all $\mathsf{Y} \in \mathcal{Y}$ and $f'(\mathsf{Aux}) \leq f(\mathsf{Aux})$ for all $\mathsf{Aux} \notin \mathcal{Y}$.

Proof: Solutions of $P_{\max}$ have to satisfy the constraints of P_{LP} because the maximum over a set is at least as large as any element in the set.

Consider a solution f of P_{LP}. If at least one constraint $\mathsf{Aux}_i \geq e_{i,j}$ for every Aux_i is tight, then f is also a solution for $P_{\max}$. Otherwise, let i be the smallest index of a variable for which no constraint is tight. Reducing $f'(\mathsf{Aux}_i)$ to $\max\{f'(e_{i,1}), \ldots, f'(e_{i,k_i})\}$ still satisfies the constraints where Aux_i occurs on the left-hand side. This is well-defined because no $e_{i,j}$ can depend on Aux_i. In all other constraints Aux_i can only occur on the right-hand side and only with a positive coefficient. Reducing the value of Aux_i

cannot make such a constraint unsatisfied if it was previously satisfied. Since Aux_i does not occur in constraints for variables $\mathsf{Aux}_{i'}$ with $i' < i$, reducing Aux_i does not affect such constraints. Therefore, after the reduction, the smallest index of a variable with loose constraints must be larger than i, and we can continue reducing values of variables Aux_i until every variable has at least one tight constraint. The modified solution is a solution of $P_{\max}$ and satisfies $f'(\mathsf{Y}) = f(\mathsf{Y})$ for all $\mathsf{Y} \in \mathcal{Y}$ and $f'(\mathsf{Aux}) \leq f(\mathsf{Aux})$ for all $\mathsf{Aux} \notin \mathcal{Y}$. □

As a corollary, we can use this result to represent a "symbolic" version of the bucket elimination algorithm with unknowns $\mathcal{Y}$ as an LP. (Note that the constraints generated by the bucket elimination algorithm have exactly the form required by the theorem if functions in Ψ produce linear expressions.) This LP has the property that for every assignment to the unknowns $\mathcal{Y}$ there exists a feasible solution, and the values of Aux_m in these feasible solutions are exactly the set of numbers greater or equal to $\mathit{Max}(\Psi)$ for the given assignment to $\mathcal{Y}$. (For simplicity, it would be preferable if *only* $\mathit{Max}(\Psi)$ itself resulted in a feasible assignment to Aux_m, but we will see that the weaker property where Aux_m may overestimate $\mathit{Max}(\Psi)$ is sufficient for our purposes.) We write the set of constraints for this LP as $P_{\mathrm{LP}(\Psi,\sigma)}$.

Corollary C.1. *The constraints in $P_{\mathrm{LP}(\Psi,\sigma)}$ are equivalent to $\mathsf{Aux}_m \geq \mathit{Max}(\Psi)$.*

An LP can be solved in time that is polynomial in its size, so to bound the complexity, we have to consider the number and size of the constraints in $P_{\mathrm{LP}(\Psi,\sigma)}$. Dechter (2003) defines the *dependency graph* of a problem Ψ over variables $\mathcal{X}$ as the undirected graph $G(\Psi) = \langle \mathcal{X}, E \rangle$ with set of vertices given by the variables $\mathcal{X}$ and an edge $\langle X, X' \rangle \in E$ iff $X \neq X'$ and there is a scoped function $\langle S, \psi \rangle$ in Ψ with $\{X, X'\} \subseteq S$. Given an undirected graph G and an order of its nodes σ, a *parent* of a node n is a neighbor of n that precedes n in σ. Dechter defines the *induced graph* of G along σ as the result of processing each node of G in descending order of σ and for each node connecting each pair of its parents if they are not already connected. The induced width of G along σ then is the maximal number of parents of a node in the induced graph of G along σ.

If there are n variables in $\mathcal{X}$ and each of their domains is bounded by d, then eliminating variable X_i adds one equation $\mathsf{Aux}_{X_i,\nu} = \max_{j \in \mathit{dom}(X_i)} e_{i,j}$ for each valuation ν of the scope S_{X_i} (variables relevant for X_i). The size of this scope is limited by the induced width $w(\sigma)$, so the number of valuations is limited by $d^{w(\sigma)}$. As there are n buckets to eliminate, the number of auxiliary variables in the LP can thus be bounded by $O(nd^{w(\sigma)})$. Each such variable occurs in $|\mathit{dom}(X_i)| \leq d$ constraints of the form $\mathsf{Aux}_{X_i,\nu} \geq e_{i,j}$ in $P_{\mathrm{LP}(\Psi,\sigma)}$, so there are $O(nd^{w(\sigma)+1})$ constraints.

Theorem C.3. *Let Ψ be a set of scoped functions over the variables in $\mathcal{X}$ that map to linear expressions over $\mathcal{Y}$. Let σ be an ordering for $\mathcal{X}$.*

Then $P_{\mathrm{LP}(\Psi,\sigma)}$ has $O(|\mathcal{Y}| + |\mathcal{X}| d^{w(\sigma)})$ variables and $O(|\mathcal{X}| d^{w(\sigma)+1})$ constraints, where $d = \max_{X \in \mathcal{X}} |\mathit{dom}(X)|$ and $w(\sigma)$ is the induced width of $G(\Psi)$.

The smallest possible induced width of $G(\Psi)$ along any order σ is called the induced width of $G(\Psi)$ and equals the treewidth of $G(\Psi)$ (Dechter, 2003). Unfortunately, finding the induced width or a minimizing order is NP-hard. However, it is fixed-parameter tractable (Downey and Fellows, 1999) with the treewidth as the parameter (Bodlaender, 1996).

Bibliography

Aghighi, M., and Jonsson, P. 2014. Oversubscription planning: Complexity and compilability. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2221–2227. AAAI Press.

Alcázar, V., and Torralba, Á. 2015. A reminder about the importance of computing and exploiting invariants in planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In Helmert, M., and Röger, G., eds., *Proceedings of the Sixth Annual Symposium on Combinatorial Search (SoCS 2013)*, 29–37. AAAI Press.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton University Press.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4(1):238–252.

Bertsekas, D. P. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific.

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In Mertsching, B.; Hund, M.; and Aziz, Z., eds., *Proceedings of the 32nd Annual German Conference on Artificial Intelligence (KI 2009)*, volume 5803 of *Lecture Notes in Artificial Intelligence*, 9–16. Springer-Verlag.

Black, M. 1946. *Critical Thinking: An Introduction to Logic and the Scientific Method*. Prentice Hall.

Bodlaender, H. L. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing* 25(6):1305–1317.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1):5–33.

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the convergence of real-time dynamic programming. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, 12–21. AAAI Press.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 329–334. IOS Press.

Bonet, B., and van den Briel, M. 2014. Flow-based heuristics for optimal planning: Landmarks and merges. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 47–55. AAAI Press.

Bonet, B. 2013. An admissible heuristic for SAS^+ planning obtained from the state equation. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2268–2274. AAAI Press.

Brafman, R. I., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 28–35. AAAI Press.

Bylander, T. 1997. A linear programming heuristic for optimal planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*, 694–699. AAAI Press.

Camm, J. D.; Raturi, A. S.; and Tsubakitani, S. 1990. Cutting big M down to size. *Interfaces* 20(5):61–66.

Chen, H., and Giménez, O. 2007. Act local, think global: Width notions for tractable planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 73–80. AAAI Press.

Chen, H., and Giménez, O. 2009. On-the-fly macros. In *Logic, Language, Information and Computation*, volume 5514 of *Lecture Notes in Computer Science*, 155–169. Springer-Verlag.

Chung, F.; Graham, R.; Morrison, J.; and Odlyzko, A. 1995. Pebbling a chessboard. *The American Mathematical Monthly* 102(2):113–123.

Ciardo, G., and Siminiceanu, R. 2002. Using edge-valued decision diagrams for symbolic generation of shortest paths. In Aagaard, M., and O'Leary, J. W., eds., *Proceedings of the Fourth International Conference on Formal Methods in Computer-Aided*

Design (FMCAD 2002), volume 2517 of *Lecture Notes in Computer Science*, 256–273. Springer-Verlag.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. A hybrid relaxed planning graph-LP heuristic for numeric planning domains. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 52–59. AAAI Press.

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44:1–96.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Dantzig, G. B., and Wolfe, P. 1960. Decomposition principle for linear programs. *Operations Research* 8(1):101–111.

Dantzig, G. B. 1951. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation*, 339–347. John Wiley & Sons.

Davies, T. O.; Pearce, A. R.; Stuckey, P.; and Lipovetzky, N. 2015. Sequencing operator counts. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 61–69. AAAI Press.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113(1):41–85.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Do, M. B., and Kambhampati, S. 2001. Sapa: A domain-independent heuristic metric temporal planner. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 57–68. AAAI Press.

Domshlak, C., and Mirkis, V. 2015. Deterministic oversubscription planning as heuristic search: Abstractions and reformulations. *Journal of Artificial Intelligence Research* 52:97–169.

Domshlak, C.; Hoffmann, J.; and Katz, M. 2015. Red-black planning: A new systematic approach to partial delete relaxation. *Artificial Intelligence* 221:73–114.

Domshlak, C.; Katz, M.; and Lefler, S. 2012. Landmark-enhanced abstraction heuristics. *Artificial Intelligence* 189:48–68.

Downey, R. G., and Fellows, M. R. 1999. *Parameterized Complexity*. Springer.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2006. Directed model checking with distance-preserving abstractions. In Valmari, A., ed., *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, 19–34. Springer-Verlag.

Dreier, J., and Kerschbaum, F. 2011. Practical privacy-preserving multiparty linear programming based on problem transformation. In *Proceedings of the 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and the 2011 IEEE Third International Conference on Social Computing (PASSAT/SocialCom 2011)*, 916–924. IEEE.

Edelkamp, S. 2001. Planning with pattern databases. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 84–90. AAAI Press.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In Edelkamp, S., and Lomuscio, A., eds., *Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006)*, 35–50.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 130–137. AAAI Press.

Fan, G.; Müller, M.; and Holte, R. 2014. Non-linear merging strategies for merge-and-shrink based on variable interactions. In Edelkamp, S., and Barták, R., eds., *Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS 2014)*, 53–61. AAAI Press.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of Artificial Intelligence Research* 22:279–318.

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

Fišer, D.; Štolba, M.; and Komenda, A. 2015. MAPlan. In *Proceedings of the First Competition of Distributed and Multi-Agent Planners (CoDMAP)*, 8–10. AAAI Press.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman.

Gerevini, A., and Schubert, L. 2001. DISCOPLAN: an efficient on-line system for computing planning domain invariants. In Cesta, A., and Borrajo, D., eds., *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, 284–288. AAAI Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Goldfarb, D.; Hao, J.; and Kai, S.-R. 1990. Efficient shortest path simplex algorithms. *Operations Research* 38(4):624–628.

Hansson, O.; Mayer, A.; and Yung, M. 1992. Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences* 63(3):207–227.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, 140–149. AAAI Press.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007)*, 1007–1012. AAAI Press.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, 1163–1168. AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M., and Mattmüller, R. 2008. Accuracy of admissible heuristic functions in selected planning domains. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 938–943. AAAI Press.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM* 61(3):16:1–63.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2008. Explicit-state abstraction: A new method for generating heuristic functions. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 1547–1550. AAAI Press.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence* 143(2):219–262.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Heusner, M.; Wehrle, M.; Pommerening, F.; and Helmert, M. 2014. Under-approximation refinement for classical planning. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 365–369. AAAI Press.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J.; Kissmann, P.; and Torralba, Á. 2014. "Distance"? Who cares? Tailoring merge-and-shrink heuristics to detect unsolvability. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 441–446. IOS Press.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research* 20:291–341.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *Journal of Artificial Intelligence Research* 24:685–758.

Holmgren, J.; Persson, J. A.; and Davidsson, P. 2009. Agent-based Dantzig-Wolfe decomposition. In Hakansson, A., and Hartung, R., eds., *Proceedings of the Third KES International Symposium (KES-AMSTA 2009)*, 754–763. Springer-Verlag.

Holte, R. C. 2010. Common misconceptions concerning heuristic search. In Felner, A., and Sturtevant, N., eds., *Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS 2010)*, 46–51. AAAI Press.

Holte, R. C. 2013. Korf's conjecture and the future of abstraction-based heuristics. In Frisch, A. M., and Gregory, P., eds., *Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation (SARA 2013)*, 128–131. AAAI Press.

Hooker, J. N., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96(1):33–60.

IBM. 2017. IBM® ILOG® CPLEX® Optimization Studio, v12.5.1. `http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud`.

Imai, T., and Fukunaga, A. 2014. A practical, integer-linear programming model for the delete-relaxation in cost-optimal planning. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 459–464. IOS Press.

Imai, T., and Fukunaga, A. 2015. On a practical, integer-linear programming model for delete-free tasks and its use as a heuristic for cost-optimal planning. *Journal of Artificial Intelligence Research* 54:631–677.

Ivankovic, F.; Haslum, P.; Thiébaux, S.; Shivashankar, V.; and Nau, D. S. 2014. Optimal planning with global numerical state constraints. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 145–153. AAAI Press.

Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* 4(4):373–395.

Karpas, E., and Domshlak, C. 2009. Cost-optimal planning with landmarks. In Boutilier, C., ed., *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1728–1733. AAAI Press.

Katz, M., and Domshlak, C. 2007a. Structural patterns heuristics: Basic idea and concrete instance. In *ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning: Progress, Ideas, Limitations, Challenges*.

Katz, M., and Domshlak, C. 2007b. Structural patterns of tractable sequentially-optimal planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 200–207. AAAI Press.

Katz, M., and Domshlak, C. 2008a. Optimal additive composition of abstraction-based admissible heuristics. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 174–181. AAAI Press.

Katz, M., and Domshlak, C. 2008b. Structural patterns heuristics via fork decomposition. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E., eds., *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008)*, 182–189. AAAI Press.

Katz, M., and Domshlak, C. 2009. Structural-pattern databases. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 186–193. AAAI Press.

Katz, M., and Domshlak, C. 2010a. Implicit abstraction heuristics. *Journal of Artificial Intelligence Research* 39:51–126.

Katz, M., and Domshlak, C. 2010b. Optimal admissible composition of abstraction heuristics. *Artificial Intelligence* 174(12–13):767–798.

Keller, T.; Pommerening, F.; Seipp, J.; Geißer, F.; and Mattmüller, R. 2016. State-dependent cost partitionings for Cartesian abstractions in classical planning. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3161–3169. AAAI Press.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2012. Semi-relaxed plan heuristics. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 128–136. AAAI Press.

Keyder, E.; Hoffmann, J.; and Haslum, P. 2014. Improving delete relaxation heuristics through explicitly represented conjunctions. *Journal of Artificial Intelligence Research* 50:487–533.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and complete landmarks for and/or graphs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, 335–340. IOS Press.

Khachiyan, L. G. 1980. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics* 20(1):53–72.

Knuth, D. E. 1992. Two notes on notation. *American Mathematical Monthly* 99(5):403–422.

Kontsevitch, M. 1981. Problem M715. *Kvant* 12(11):21–22. (In Russian).

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1–2):9–22.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening A*. *Artificial Intelligence* 129:199–218.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Korf, R. E. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*, 700–705. AAAI Press.

Korte, B., and Vygen, J. 2001. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2nd edition.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In De Raedt, L.; Bessiere, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P., eds., *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 540–545. IOS Press.

Lipovetzky, N.; Muise, C.; and Geffner, H. 2016. Traps, invariants, and dead-ends. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 211–215. AAAI Press.

Lolli, G. 1763. *Osservazioni teorico-pratiche sopra il giuoco degli scacchi*. Bologna: Stamperia di S. Tommaso D'Aquino. (In Italian).

Lougee-Heimer, R. 2003. The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development* 47(1):57–66.

Mangasarian, O. L. 2011. Privacy-preserving linear programming. *Optimization Letters* 5(1):165–172.

Mausam, and Kolobov, A. 2012. *Planning with Markov Decision Processes*. Morgan & Claypool.

McCarthy, J. 1964. A tough nut for proof procedures. Stanford Artificial Intelligence Project, Memo 16.

McDermott, D. 1999. Using regression-match graphs to control search in planning. *Artificial Intelligence* 109(1–2):111–159.

Muise, C., and Lipovetzky, N. 2014. First Unsolvability International Planning Competition (UIPC 2014). `http://unsolve-ipc.eng.unimelb.edu.au`. Accessed March 2017.

Muise, C. 2016. Planning.Domains. System Demonstrations on the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016) `http://www.haz.ca/papers/planning-domains-icaps16.pdf`. Accessed Dezember 2016.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for parallel and distributed systems. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, 1265–1266. IFAAMAS/ACM.

Numberphile. 2013. Pebbling a chessboard. `https://www.youtube.com/watch?v=lFQGSGsXbXE`. (Accessed Feb. 2016).

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pommerening, F., and Helmert, M. 2015. A normal form for classical planning tasks. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 188–192. AAAI Press.

Pommerening, F., and Seipp, J. 2016. Fast Downward dead-end pattern database. In Muise, C., and Lipovetzky, N., eds., *Unsolvability International Planning Competition: planner abstracts*, 2.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2014a. From non-negative to general operator cost partitioning: Proof details. Technical Report CS-2014-005, University of Basel, Department of Mathematics and Computer Science.

Pommerening, F.; Röger, G.; Helmert, M.; and Bonet, B. 2014b. LP-based heuristics for cost-optimal planning. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 226–234. AAAI Press.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

Pommerening, F.; Helmert, M.; and Bonet, B. 2017a. Abstraction heuristics, cost partitioning and network flows. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 228–232. AAAI Press.

Pommerening, F.; Helmert, M.; and Bonet, B. 2017b. Higher-dimensional potential heuristics for optimal classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3636–3643. AAAI Press.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In Rossi, F., ed., *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2357–2364. AAAI Press.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Rintanen, J. 2000. An iterative algorithm for synthesizing invariants. In Kautz, H., and Porter, B., eds., *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, 806–811. AAAI Press.

Rintanen, J. 2017. Schematic invariants by reduction to ground invariants. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3644–3650. AAAI Press.

Röger, G., and Pommerening, F. 2015. Linear programming for heuristics in optimal planning. In *AAAI 2015 Workshop on Planning, Search, and Optimization*, 69–76.

Röger, G.; Pommerening, F.; and Helmert, M. 2014. Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 765–770. IOS Press.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence — A Modern Approach*. Prentice Hall.

Scherrer, S.; Pommerening, F.; and Wehrle, M. 2015. Improved pattern selection for PDB heuristics in classical planning (extended abstract). In Lelis, L., and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 216–217. AAAI Press.

Schrijver, A. 1998. *Theory of linear and integer programming*. John Wiley & Sons.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351. AAAI Press.

Seipp, J., and Helmert, M. 2014. Diverse and additive Cartesian abstraction heuristics. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 289–297. AAAI Press.

Seipp, J.; Pommerening, F.; Röger, G.; and Helmert, M. 2016a. Correlation complexity of classical planning domains. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3242–3250. AAAI Press.

Seipp, J.; Pommerening, F.; Sievers, S.; Wehrle, M.; Fawcett, C.; and Alkhazraji, Y. 2016b. Fast Downward Aidos. In Muise, C., and Lipovetzky, N., eds., *Unsolvability International Planning Competition: planner abstracts*, 28–38.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. downward-lab 2.0. `https://doi.org/10.5281/zenodo.399255`.

Seipp, J.; Keller, T.; and Helmert, M. 2017a. A comparison of cost partitioning algorithms for optimal classical planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 259–268. AAAI Press.

Seipp, J.; Keller, T.; and Helmert, M. 2017b. Narrowing the gap between saturated and optimal cost partitioning for classical planning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI 2017)*, 3651–3657. AAAI Press.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New optimization functions for potential heuristics. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 193–201. AAAI Press.

Shannon, C. E. 1950. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41(314):256–275.

Sievers, S.; Wehrle, M.; and Helmert, M. 2014. Generalized label reduction for merge-and-shrink heuristics. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2014)*, 2358–2366. AAAI Press.

Sievers, S.; Wehrle, M.; and Helmert, M. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 294–298. AAAI Press.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International*

Conference on Automated Planning and Scheduling (ICAPS 2004), 393–401. AAAI Press.

Steinmetz, M., and Hoffmann, J. 2016. Towards clause-learning state space search: Learning to recognize dead-ends. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)*, 760–768. AAAI Press.

Štolba, M.; Fišer, D.; and Komenda, A. 2016. Potential heuristics for multi-agent planning. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 308–316. AAAI Press.

Teichteil-Königsbuch, F.; Kuter, U.; and Infantes, G. 2010. Incremental plan aggregation for generating policies in MDPs. In *Proceedings of the Nineth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2010)*, 1231–1238. IFAAMAS/ACM.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2016. Abstraction heuristics for symbolic bidirectional search. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3272–3278. AAAI Press.

Trevizan, F.; Thiébaux, S.; and Haslum, P. 2017. Occupation measure heuristics for probabilistic planning. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, 306–315. AAAI Press.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-based heuristic for optimal planning. In Bessiere, C., ed., *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *Lecture Notes in Computer Science*, 651–665. Springer-Verlag.

Wilt, C., and Ruml, W. 2015. Building a heuristic for greedy search. In Lelis, L., and Stern, R., eds., *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*, 131–139. AAAI Press.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research* 32:631–662.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In Boddy, M.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*, 352–359. AAAI Press.

Zhu, L., and Givan, R. 2003. Landmark extraction via planning graph propagation. In *ICAPS 2003 Doctoral Consortium*, 156–160.