Dissertation

On Planning with State-dependent Action Costs

Florian Geißer

2018

Faculty of Engineering
Albert-Ludwigs-Universität Freiburg im Breisgau

Dean:
Prof. Dr. Hannah Bast, *University of Freiburg, Germany*

PhD advisor and first reviewer:
Prof. Dr. Bernhard Nebel, *University of Freiburg, Germany*

Second reviewer:
Prof. Dr. Bernd Becker, *University of Freiburg, Germany*

Date of defense:
January 14th, 2019

Abstract

Automated Planning is an established research field of Artificial Intelligence. While in probabilistic planning models, such as Stochastic Shortest Path problems, the cost of an action can be *state-dependent*, the classical deterministic planning literature mostly considers the cost of actions to be constant. Therefore, even if a planning task naturally admits state-dependent costs, the modeler has to distribute these costs over multiple copies of the action. This does not only introduce additional burden on the modeler, but it also hides structure which may be apparent in the action cost function and may provide useful information for planning algorithms.

In this thesis we do away this restriction to constant costs, by considering *classical planning with state-dependent action costs*. We show how we can make use of *edge-valued multi-valued decision diagrams* (EVMDDs) to represent the action cost functions and provide *compilations* of state-dependent action cost tasks to classical tasks with constant costs, which allows us to leverage classical planning tools. These compilations are polynomial in the size of the underlying EVMDDs. While their size is worst-case exponential, for many commonly encountered cost functions this results in a classical planning task with compact size.

Heuristic search is one of the most prominent tools in classical planning to produce optimal solutions. Two well-known families of heuristics are *delete relaxation* and *abstraction heuristics*. We generalize both families to planning with state-dependent action costs and show how we can use the EVMDD representation to efficiently compute the generalized heuristics. Furthermore, we provide a theoretical analysis of our introduced compilations, showing that many heuristics are *invariant under compilation*, i.e. the compilation preserves the heuristic estimates and does not lead to a loss of information. We empirically evaluate how these theoretical results behave in practice, by comparing different compilations and heuristics on a benchmark set consisting of tasks with state-dependent action costs.

Zusammenfassung

Handlungsplanung ist ein etablierter Forschungsbereich der künstlichen Intelligenz. Während in probabilistischen Planungsmodellen, wie beispielsweise in “Stochastic Shortest Path” Problemen, die Kosten einer Aktion *zustandsabhängig* sein können, wird in der klassischen Planungsliteratur meist von konstanten Aktionskosten ausgegangen. Selbst wenn ein Planungsproblem von Natur aus zustandsabhängige Aktionskosten beinhaltet muss der Modellierer des Problems somit diese Kosten über mehrere Kopien der ursprünglichen Aktion verteilen. Das hat nicht nur einen erhöhten Aufwand für den Modellierer zur Folge, sondern dadurch wird auch die Struktur der ursprünglichen Kostenfunktion verborgen, die jedoch nützliche Informationen für Planungsalgorithmen liefern könnte.

In dieser Dissertation beseitigen wir diese Einschränkung auf konstante Aktionskosten, indem wir *klassische Handlungsplanung mit zustandsabhängigen Aktionskosten* betrachten. Wir zeigen, wie wir von *kantengewichteten mehrwertigen Entscheidungsdiagrammen* (“edge-valued multi-valued decision diagrams”, kurz EVMDDs) Gebrauch machen können um die Kostenfunktionen darzustellen. Basierend auf dieser Darstellung liefern wir *Kompilierungen*, mit denen ein Planungsproblem mit zustandsabhängigen Kosten in ein klassisches Planungsproblem mit konstanten Kosten transformiert werden kann, was es uns erlaubt etablierte klassische Planungsalgorithmen zu verwenden. Die Größe dieser Kompilierungen ist polynomiell in der Größe der entsprechenden Entscheidungsdiagramme. Dies führt zwar im schlimmsten Fall zu exponentiell großen Planungsproblemen, für viele übliche Kostenfunktionen ist die Größe jedoch kompakt.

Um Planungsprobleme optimal zu lösen ist *heuristische Suche* eine der meistgenutzten Techniken der klassischen Handlungsplanung. Zwei weit verbreitete Heuristikfamilien sind die sogenannte *Delete Relaxierung*, sowie *Abstraktionsheuristiken*. Wir generalisieren diese beiden Familen auf Planungsprobleme mit zustandabhängigen Aktionskosten und zeigen wie man die Darstellung der Kostenfunktion als EVMDD nutzen kann um diese Generalisie-

rung effizient zu berechnen. Außerdem liefern wir eine theoretische Analyse unserer eingeführten Kompilierungen. Wir zeigen, dass viele dieser Heuristiken *invariant unter Kompilierung* sind, das heißt die Heuristik erhält ihre Werte in der Kompilierung und wir verlieren durch den Kompilierungsprozess keine Informationen bezüglich der Heuristik. In einer empirischen Untersuchung bewerten wir unsere theoretischen Resultate in der Praxis. Wir vergleichen verschiedene Heuristiken unter verschiedenen Kompilierungen auf einem Datensatz, der aus mehreren Planungsproblemen mit zustandsabhängigen Aktionskosten besteht.

Acknowledgments

In the years during my doctoral process (and before!) there have been numerous people who contributed along the way, up to this final outcome and I want to use this space to thank these fantastic people. First and foremost, I want to thank my advisor Bernhard Nebel, who not only offered me a position in his wonderful research group five years ago, but who also made it possible for me to meet various people around the world, be it by attending international conferences or by visiting foreign research groups. These interactions led to valuable input by multiple researchers. Additionally, Bernhard gave me the freedom to pursue my own ideas which led to the research presented in this thesis. Finally, without the established relaxed and productive atmosphere in his research group I certainly would not have had the endurance required to finish this thesis over the last month.

Next, I want to express my deepest gratitude to my friend and colleague Robert Mattmüller, who accompanied my research beginning from my Master's degree up to this moment. Robert teached me to appreciate the beauty of formal correctness and without him many results given in this thesis would not have been possible. It has been (and hopefully will continue to be) a pleasure to work with him in all these years.

While Robert contributed very much to my proficiency in formal correctness, Thomas Keller guided me over the years to increase my proficiency with the C++ programming language, by inviting me five years ago to work with him on the PROST planner. This has been a fun and fascinating journey and I'm glad the long number of issue tickets we still have to process indicates that we will continue to work together.

Of course I also want to thank all my other colleagues for making our group such a great place to work and giving rise to compelling scientific and casual discussions. Thank you Johannes Aldinger, Thorsten Engesser, Felix Lindner, Tim Schulte, David Speck, Benedict Wright, and all the other colleagues who left our group over the last couple of years, especially Christian Dornhege, Andreas Hertle and Manuela Ortlieb for a continued friendship

over the years, and Dali Sun for the years of work we had together on the Karis system. Finally, I also want to thank Ulrich Jakob for his continued work in keeping our infrastructure running and Petra Geiger for handling all the amount of office work which comes up during so many years.

I am also very grateful to Thomas Keller, Robert Mattmüller and David Speck for proofreading my thesis while being limited by my extremely sportive timeline. Especially Robert has to be pointed out, who spent quite some time and has made me aware of many minor and major issues.

Another group to which I have to express my gratitude is the planning group of Malte Helmert in Basel. Our joint reading group has given rise to many exciting discussions and one could be sure that whenever attending any conference there would be someone from either Freiburg or from Basel to talk to.

Next, I want to thank Sylvie Thiébaux for providing me the opportunity to work in her group and continuing AI research on the other side of the world!

In the end, work is not all that counts, and luckily there have been multiple people enriching my life in other parts than AI. Therefore, I thank my friends for supporting me over the years and allowing distractions from the daily work life. First, thank you Tim Schulte and Robert Grönsfeld, the two people which remained in Freiburg from our group of bachelor students meeting more than 10 years ago. Of course I do not want to miss out our other friends: Kim Bischof, Mirko Brodesser, André Doser, Philipp Lerche and Kyanoush Seyed Yahosseini (cwh!). Hundreds of miles distance have not undermined our friendship, so let's keep it this way and just raise the scale by an order of magnitude.

Zu guter Letzt möchte ich meiner Familie danken, insbesondere meinen Eltern Karl-Heinz und Jutta. Ohne ihre andauernde Unterstützung wäre dies alles nicht möglich gewesen und ihre Hilfe war insbesondere in den letzten von Umzugstress geplagten Tagen von unschätzbarem Wert. Ein Sohn kann sich keine bessere Familie wünschen.

Thank you!

Contents

CHAPTER 1

Introduction

Reasoning, *planning* and *problem solving* are all conceived to be parts of what is generally understood as *intelligence*. In the context of Artificial Intelligence, the related established research field is Automated Planning (or AI planning), where an agent has to reach a given goal by applying a sequence of actions (Russel and Norvig 2010). While modern applications, such as high-level planning for socially interacting robots (Petrick and Foster 2013) or behaviour planning of AI agents in video games (Orkin 2006) fall clearly in the category of AI planning, the field also includes applications which are not immediately considered as planning problems, such as greenhouse logistics (Helmert and Lasinger 2010), organic synthesis (Masoumi et al. 2015), or penetration testing (Hoffmann 2015).

Particularly useful is the sub-field of *domain-independent* planning, which explores general techniques which do not rely on domain-dependent knowledge and can thus be applied to a variety of applications. Most often, these application domains can be modeled in an appropriate planning language: the *planning domain definition language* (McDermott et al. 1998) allows to model *fully observable*, *deterministic*, *static* planning problems. These problems are generally considered as *classical planning problems*. Over the years, the language has received multiple extensions, which allow, for example, expressing *conditional* (or *state-dependent*) effects. If the problems are non-deterministic, the *probabilistic planning domain definition language* (PPDDL) (Younes and Littman 2004) empowers the user to reflect the probabilistic nature of the problem in the model. The *relational dynamic influence diagram language* (Sanner 2010) is another formalism, which gives the modeler even more leeway in the ability to model their task by allowing more complex expressions than PPDDL permits. The tasks modeled with these languages are often understood as factored *Markov Decision Processes* (MDPs).

While the probabilistic aspect most prominently distinguishes a classical planning problem from an MDP, there is another, more subtle, difference: in many probabilistic planning tasks (for example, in the problems of the Inter-

national Probablistic Planning Competition), rewards (or costs) obtained after the application of actions are often *state-dependent*. While the focus of probabilistic planning is usually the probabilistic nature of the problem and not the state-dependent aspect, the latter still remains if we disregard the probabilistic portion. Therefore, we remain with a classical planning problem, but with state-dependent action costs.

In classical planning, however, action costs are normally considered to be constant. As a consequence, naturally arising state-dependent costs, such as the fuel consumption based on the distance to reach another location or the increased fuel-requirement of a vehicle depending on its load, have to be distributed over multiple copies of the original action, instead of being represented as a state-dependent cost function. This does not only increase the burden on the modeler and allow additional room for errors, but it also conceals *structure*, which may be present in the action cost function and could be exploited algorithmically. In this thesis, we do away with the restriction of classical planning to constant costs, by considering *classical planning tasks with state-dependent action cost*. Therefore, our setting is fully observable, static, and most importantly also deterministic. Moreover, we are mostly concerned with *optimal* planning, i.e. we must guarantee that the solution minimizes the cost.

Classical planning is theoretically (and practically) well-understood (Bäckström and Nebel 1995; Bylander 1994): in general, finding any solution is already PSPACE-hard, as is finding an optimal solution. Nevertheless, many different techniques have been successfully employed on a variety of domains. One of the most prominent techniques to obtain a satisficing or even optimal solution is *heuristic search*, where a distance estimator, the heuristic, guides the search towards a solution. Over the years, different heuristics have been developed, often excelling in some, but never in all domains. However, all of these heuristic are only defined for tasks with constant cost actions. The question is therefore: *can we generalize classical planning heuristics to tasks with state-dependent action cost* and *can we make use of established classical planning tools to solve such tasks*. To answer this question, we discuss how we can *represent* state-dependent action costs, such that we can exhibit and exploit the structure of the cost function. This representation allows us on one hand to generalize two well-known classes of heuristics: *delete relaxation heuristics* (Bonet et al. 1997; McDermott 1996) and *Cartesian abstraction heuristics* (Seipp and Helmert 2018), which include *pattern database heuristics* (Culberson and Schaeffer 1998; Edelkamp 2001). On the other hand, it allows us to come up with different compilation schemes, which transforms a state-dependent action cost task to a classical planning task, while preserving heuristic accuracy.

1.1 Contributions

This thesis makes the following contributions to the field of AI planning:

- Compiling a task with state-dependent action cost to a classical planning task with constant cost allows to use established classical planning tools to generate solutions for such problems. A straightforward compilation of state-dependent action costs has exponential overhead. We present alternative compilations that make use of edge-valued multi-valued decision diagrams (EVMDDs), which symbolically represent the action cost function. This compilation preserves optimal plan costs and is polynomial in the size of the decision diagrams. While in the worst-case, the size of these diagrams grows exponentially, size only grows polynomially for many common types of cost functions.

- Delete relaxation heuristics are a well-known family of heuristics used to guide search. We generalize the optimal delete-relaxation heuristic h^+, as well as two approximations: the additive heuristic h^{add} and the maximum heuristic h^{max}. We present a theoretical analysis, which shows that the decision diagrams representing the cost functions can be used to efficiently evaluate action costs for Cartesian sets of states, which includes relaxed states. Furthermore, we show that h^+ and h^{add} are invariant under the presented compilations, i.e. the estimate obtained from the generalized heuristic is preserved in the compiled task. For h^{max}, we show that we can achieve invariance if we consider a certain class of decision diagrams. These results allow the use of these heuristics with a classical planner to solve tasks with state-dependent action costs.

- Abstraction heuristics are often more powerful than delete relaxation heuristics and are still considered as some of the strongest heuristics the classical planning literature has to offer. In our theoretical analysis we generalize abstraction heuristics to tasks with state-dependent action costs and show how we can efficiently compute abstract transition weights for Cartesian abstractions.

  We show that any type of abstraction heuristic is invariant under exponential compilation. For the compilations based on decision diagrams, Cartesian abstraction heuristics are invariant under these compilations, if non-deterministic transitions in the abstraction do not have different weights and if the abstraction preserves variables introduced by the compilation. This generalizes a previous result which has shown invariance for Cartesian abstractions with non-deterministic transitions. As projection abstractions never induce non-deterministic transitions, we can carry over this result to pattern database heuristics. We show

how to achieve invariance for the canonical pattern database heuristic and discuss how to adapt the h^{iPDB} approach to make use of this result. Therefore, we have an efficient compilation which yields strong admissible heuristic estimates based on pattern database heuristics.

For arbitrary Cartesian abstractions, we show that the compilation still provides informative admissible estimates. We discuss how we can generalize the counterexample-guided abstraction refinement algorithm to include state-dependent action costs, by introducing cost divergence as a fourth type of flaw. This result allows the generation of Cartesian abstractions for tasks with state-dependent action costs.

- Finally, we provide an empirical evaluation analyzing the aforementioned theoretical results in practice. We modified the Fast Downward planner (Helmert 2006b) to be able to deal with state-dependent action costs and implemented the different compilation schemes. This allows us to compare the interaction between different compilations and heuristics on a benchmark set consisting of several tasks with state-dependent action costs. We analyze the heuristic accuracy, as well as the overall performance for different configurations of heuristic and compilation.

1.2 Outline

The thesis is structured as follows: in Chapter 2 we introduce the preliminaries, specifically a *formal definition of planning tasks with state-dependent action costs*. We discuss how we can model such tasks and how to use decision diagrams to symbolically represent the action cost function. We also give some brief results regarding the interaction of edge-valued decision diagrams and Cartesian sets of states. Finally, we introduce different compilations which allow the transformation to a classical planning task.

Chapter 3 introduces and generalizes *delete relaxation* heuristics. In particular, we generalize the optimal delete relaxation heuristic, as well as two approximations: the additive and the maximum heuristic. We theoretically evaluate the different compilations in terms of heuristic invariance for all three heuristics.

We then introduce *abstraction heuristics* in Chapter 4. Once again, we generalize the definition of abstraction heuristics to tasks with state-dependent action costs. We first analyze *projection abstractions* and discuss how we can adapt the iterative hill-climbing search for pattern collections. The second part of this chapter deals with *Cartesian abstractions* and we show how we can come up with Cartesian abstractions for our tasks.

While all of the aforementioned chapters provide only theoretical insights, Chapter 5 provides an empirical evaluation of the different compilations we

proposed. We analyze if the theoretical results are reflected in practical performance by evaluating a benchmark set consisting of several tasks with state-dependent action costs.

We conclude the thesis by providing further discussion and related work in Chapter 6 and give a final summary of the obtained results in Chapter 7.

1.3 Relation to Published Work

Many of the results given in this thesis are published in proceedings of significant AI and automated planning conferences. In particular, the following two papers provide the backbone of this thesis:

- Florian Geißer, Thomas Keller, and Robert Mattmüller (2015). "Delete Relaxations for Planning with State-Dependent Action Costs". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. Ed. by Qiang Yang and Michael Wooldridge. AAAI Press, pp. 1573–1579

  We formally introduce planning with state-dependent action costs and generalize the additive delete relaxation heuristic to such tasks. We also present the EVMDD-based compilation and show that the additive heuristic is invariant under this compilation. This paper forms the basis for the different compilations introduced in Chapter 2, as well as for most of the theoretical results we give for the additive heuristic in Chapter 3.

- Florian Geißer, Thomas Keller, and Robert Mattmüller (2016). "Abstractions for Planning with State-Dependent Action Costs". In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*. Ed. by Amanda Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner. AAAI Press, pp. 140–148

  We define abstract planning tasks with state-dependent action costs and discuss the evaluation of cost functions in abstract states. We show that EVMDDs allow an efficient computation of abstract cost values if the abstraction is Cartesian. We also generalize the CEGAR approach to planning tasks with state-dependent action costs. The paper introduces the local minimization of EVMDDs for Cartesian states, which we introduce in Chapter 2 and also includes many of the results we give in the second part of Chapter 4.

The following papers also discuss state-dependent action costs, but have a different focus than this thesis:

- Thomas Keller, Florian Pommerening, Jendrik Seipp, Florian Geißer, and Robert Mattmüller (2016). "State-dependent Cost Partitionings for Cartesian Abstractions in Classical Planning". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*. Ed. by Subbarao Kambhampati. AAAI Press, pp. 3161–3169

  We generalize general cost-partitioning to state-dependent general cost partitioning, which dominates the state-independent version. We also take a look at saturated cost partitioning, and define its state-dependent counterpart. This allows the combination of different abstraction heuristics generated by the CEGAR algorithm to form admissible estimates. We evaluate the performance of the state-dependent saturated cost partitioning on classical planning tasks.

- Robert Mattmüller, Florian Geißer, Benedict Wright, and Bernhard Nebel (2018). "On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*. AAAI Press, pp. 6237–6245

  We consider the interaction between state-dependent action costs and conditional effects. We show that independently handling both leads to a loss of information. We then generalize EVMDDs to allow functions defined over different monoids, which allows us to represent conditional costs and effects in a single decision diagram, which can be efficiently used to compute conditional costs and effects for arbitrary Cartesian states. The EVMDD library used in the empirical evaluation given in Chapter 5 was developed during this work.

- David Speck, Florian Geißer, and Robert Mattmüller (2018). "Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams". In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. Ed. by Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs Spaan. AAAI Press

  We study EVMDD-based symbolic search for optimal planning. In particular, we define EVMDD-based representations of symbolic states and transition relations, and introduce and evaluate an EVMDD-based symbolic version of the $A^\star$ algorithm. The benchmark set for tasks with state-dependent action costs is the benchmark set we also use for the evaluation in Chapter 5.

The following papers have been published during my doctoral process, but are not related to classical planning or planning with state-dependent action costs per se:

- Florian Geißer, Thomas Keller, and Robert Mattmüller (2014). "Past, Present, and Future: An Optimal Online Algorithm for Single-Player GDL-II Games." In: *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. Ed. by Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan. IOS Press, pp. 357–362

- Thomas Keller and Florian Geißer (2015). "Better Be Lucky Than Good: Exceeding Expectations in MDP Evaluation". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, pp. 3540–3547

- Dali Sun, Florian Geißer, and Bernhard Nebel (2016). "Towards effective localization in dynamic environments". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016*, pp. 4517–4523

Finally, I also contributed to the probabilistic planning system PROST (Keller and Eyerich 2012), which provided the initial spark to take a deeper look into planning with state-dependent costs.

- Together with Thomas Keller, we submitted the PROST planner to the 5th International Probabilistic Planning Competition (IPPC 2014).

  PROST was the **winner of the Boolean MDP Track**.

- In 2018, together with David Speck, the PROST planner was modified to include a heuristic based on symbolic search with ADDs. We submitted this planner (dubbed PROST-DD) to the 6th International Probabilistic Planning Competition (IPPC 2018).

  PROST-DD was the **winner of the discrete MDP track**.

CHAPTER 2

Planning with State-Dependent Costs

2.1 Planning Tasks

In AI planning, we are interested in automatically finding a solution (a plan) for a given problem (a planning task). In this chapter, we formally define the different pieces that form a planning task. To illustrate the following definitions, we will use a planning task from the class of logistics problems, depicted in Figure 2.1. In this example task, we have a single truck, located initially at position A, which has to transport packages from A and B to C. The cost to drive between locations increases with increasing truck load.

There are many ways to model such tasks, but in this thesis we will rely on a variation of the SAS$^+$ framework (Bäckström and Nebel 1995), where states and conditions are represented by sets of variable-value pairs.

For the following definitions, we assume that $\mathcal{V}$ is a set of finite-domain variables, that is, each variable $v \in \mathcal{V}$ is associated with a finite domain

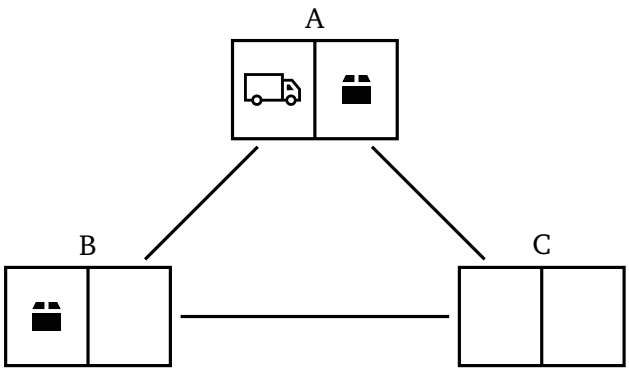


Figure 2.1: Logistics planning task. Two packages are at locations A and B and have to be transported to location C. Initially, the truck is at location A.

$\mathcal{D}_v = \{0, \ldots, |\mathcal{D}_v|-1\}$. If $\mathcal{D}_v = \{0, 1\}$, we call v a *binary variable*. For some examples, we will use variables where the domain is specified by object names instead of integer domain values, e.g. the position of the truck might be A, B or C. Formally, such a variable would have domain $\{0, 1, 2\}$, and we only use named domain values for illustration.

Definition 1 (Fact). A *fact* is a pair (v, d), alternatively written as $(v \doteq d)$, where $v \in \mathcal{V}$ and $d \in \mathcal{D}$.

A fact expresses that some variable is assigned to some particular domain value. In our logistics problem, we have the variables $\mathrm{t\text{-}at}$ with $\mathcal{D}_{\mathrm{t\text{-}at}} = \{A, B, C\}$ and $\mathrm{p_1\text{-}at}$ and $\mathrm{p_2\text{-}at}$ with domain $\{A,B,C,t\}$, which indicate the position of the truck and the packages, respectively.

With sets of facts, i.e. a mapping of variables to their domain values, we can express states and conditions.

Definition 2 (Partial variable assignments and states). A *partial variable assignment* is a function $s : \mathcal{V} \to \bigcup_{v \in \mathcal{V}} \mathcal{D}_v$, such that $s(v) \in \mathcal{D}_v$ for all $v \in \mathcal{V}$. Given a partial variable assignment s, we denote the variables for which s is defined as $\mathrm{vars}(s)$. If s assigns a value to each $v \in \mathcal{V}$, s is called a *state*. The set of all states is denoted as $\mathcal{S}$.

We often denote (partial) states with a set representation, i.e. a partial variable assignment is represented by a consistent set of facts. Sometimes, we also use a logical representation. For example, we denote the initial state of our logistics task with $(\mathrm{t\text{-}at} \doteq A) \wedge (\mathrm{p_1\text{-}at} \doteq A) \wedge (\mathrm{p_2\text{-}at} \doteq B)$, and the goal is represented by the partial state $(\mathrm{p_1\text{-}at} \doteq C) \wedge (\mathrm{p_2\text{-}at} \doteq C)$.

Throughout the thesis we will often consider *Cartesian* sets of states, which have a property which later turns out to be beneficial in our context. We use the definition given by Seipp and Helmert (2018).

Definition 3 (Cartesian sets). A set of states s_C over variables $v_1, \ldots, v_n$ is called *Cartesian* if it is of the form $D_1 \times \ldots \times D_n$, where $D_i \subseteq \mathcal{D}_{v_i}$ for $i \in \{1, \ldots, n\}$. Given a state $s \in \mathcal{S}$, we say that s_C *subsumes* s, if $s \in s_C$. Furthermore, we write $s_C(v_i)$ for D_i.

Figure 2.2 gives an example of different Cartesian and non-Cartesian sets of states. A set of states is Cartesian if we can express it as a Cartesian product of domain values. Let us for the moment assume we only have two variables

$\mathrm{p}_1\text{-at}$ and $\mathrm{p}_2\text{-at}$. Figure 2.2a depicts the Cartesian sets

$$\begin{aligned} s^1_C = \quad & \{A\} \times \{A, B\} \\ = \quad & \{(\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq A), (\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq B)\}, \\ s^2_C = \quad & \{C, t\} \times \{C, t\} \\ = \quad & \{(\mathrm{p}_1\text{-at} \doteq C) \wedge (\mathrm{p}_2\text{-at} \doteq C), (\mathrm{p}_1\text{-at} \doteq t) \wedge (\mathrm{p}_2\text{-at} \doteq t), \\ & (\mathrm{p}_1\text{-at} \doteq C) \wedge (\mathrm{p}_2\text{-at} \doteq t), (\mathrm{p}_1\text{-at} \doteq t) \wedge (\mathrm{p}_2\text{-at} \doteq C). \end{aligned}$$

On the other hand, Figure 2.2b depicts the set $\{(\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq B), (\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq A)\}$, which we can't express as a Cartesian product between domain values. Intuitively, if we consider the domain of each variable as a separate coordinate axis and states as entries in the resulting n-dimensional matrix, then a set of states is Cartesian if we can form an n-dimensional rectangle around the states (after rearranging columns/rows if necessary).

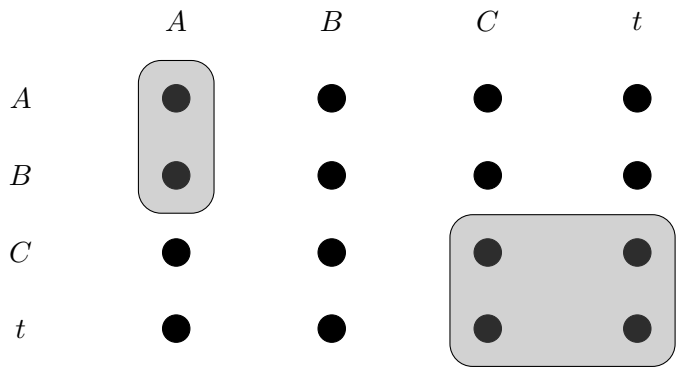
Seipp and Helmert (2018) show that the intersection of Cartesian sets is Cartesian, and the set of goal states is also Cartesian. Additionally, while a Cartesian set of states can subsume exponentially many states it can still be represented compactly, by only storing the domain values of the set.

The previous definitions allow us to model representations of the world. However, we still need ways to manipulate states of the world, i.e. actions which transform one state of the world into another state.

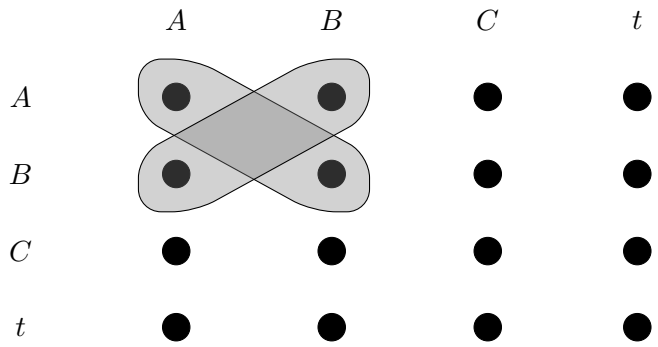
Definition 4 (Action). An *action* is a pair $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ of partial variable assignments (or: sets of facts), called *preconditions* and *effects*. By $\mathrm{pre}(a)$ we refer to the precondition of a. Similarly, $\mathrm{eff}(a)$ refers to the effect of a.

An action a is applicable in state s iff $\mathrm{pre} \subseteq s$. Applying a to s yields the state s' with $s'(v) = \mathrm{eff}(a)(v)$ where $\mathrm{eff}(a)(v)$ is defined, and $s'(v) = s(v)$ otherwise. We write $s[a]$ for s'. Given a Cartesian set of states s_C, we say that a is applicable in s_C if there exists a state $s' \in s_C$ such that a is applicable in s'. Note that the set of states in which a is applicable is also Cartesian (Seipp and Helmert 2018).

We have two kinds of actions in our logistics problem. Actions which allow us to drive the truck to a different location, and load and unload actions, which allow us to load and unload packages to and from the truck. For example, the drive action $\text{drive-AB} = \langle (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle$ requires the truck to be at location A, and after application of the action, the truck is located at location B. Similarly, we have actions drive-BA, drive-AC, drive-CA, drive-BC and drive-CB. For the load actions, we have $\text{load-p}_1 = \langle (\text{t-at} \doteq A) \wedge (\mathrm{p}_1\text{-at} \doteq A), (\mathrm{p}_1\text{-at} \doteq t) \rangle$ which allows to load package p_1 into the truck, if the truck and the package are located at A. Similarly, we have load-p_2, which allows to load package p_2 at location B. Finally, actions unload-p_1 and unload-p_2 allow



(a) Example for Cartesian sets of states.



(b) Example for non-Cartesian sets of states.

Figure 2.2

us to unload a package at location C, if the package is in the truck and the truck is located at C.

With these definitions, we can already define classical planning tasks, where actions do not have an associated cost. In this thesis, however, we are concerned with tasks with an underlying cost function on the actions.

Definition 5 (Action cost function). The *action cost function* $c_a : \mathcal{S} \to \mathbb{Q}^+$ of action a specifies the non-negative cost of applying action a in state s. The set of actions $\mathcal{A}$ induces a *global cost function* $c : \mathcal{A} \times \mathcal{S} \to \mathbb{Q}^+$, such that $c(a, s) = c_a(s)$ for all $s \in \mathcal{S}$. If it is clear from the context we will also simply write cost function instead of action cost function.

In general, this definition allows us to specify arbitrary cost functions, but allowing arbitrary cost functions can easily lead to undecidability. For example, the problem whether a Diophantine equation has an integer solution is undecidable (Cutland 1980). We could define $c(a, s) = 1$ if some Diophantine equation based on s has an integer solution. We could also define $c(a, s) = 1$ if there is a solution for a planning problem beginning in s, essentially defining

a whole planning task in the cost function. In this thesis, we will focus on cost functions we can evaluate in polynomial time (given a state) and which are typically specified as a function term, e.g. a multivariate polynomial. More formally, we distinguish three different cases:

Definition 6 (Cost function term). Let c_a be a cost function. Then c_a is either:

1. A constant function, i.e. $c_a(s) = q$, for $q \in \mathbb{Q}^+, s \in \mathcal{S}$.
2. A function $c_a(s) = s(v)$, for $s \in \mathcal{S}, v \in \mathcal{V}$, i.e. the result of c_a is the domain value of v in s. We will often write $c_a = v$ for such functions.
3. The result of an operator $\circ : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ applied to two functions $f, g : \mathcal{S} \to \mathbb{Q}$.

Such a cost function may only depend on a subset of state variables, called the *support* of c_a, which we denote by $\mathrm{vars}(c_a) = \{v_1^a, \ldots, v_k^a\}$. It will then be convenient to think of c_a as a function $c_a : \mathcal{D}_{v_1^a} \times \ldots \times \mathcal{D}_{v_k^a} \to \mathbb{Q}^+$. More formally, $\mathrm{vars}(c_a) = \{v | \exists s, s' \in \mathcal{S} \text{ s.t. } s(v) \neq s'(v) \text{ and } c_a(s) \neq c_a(s')\}$. Note that with this definition there may be variables in a function term which do not contribute to the support, e.g. the term $y + x - x$ has only y as support. We will be able to compute the support by relying on a canonical representation of cost functions, cf. Section 2.3.

For our logistics problem, we have a constant cost of 1 for the load and unload actions. The cost of the drive actions is state-dependent: the more packages loaded in the truck, the higher the cost to drive. The cost of all drive actions is specified by the term $c_{\mathrm{drive}}(s) = [\mathrm{p_1\text{-}at}(s) = t] + [\mathrm{p_2\text{-}at}(s) = t] + 1$. For clarity, we surround subterms with the Iverson bracket operator $[\cdot]$ (Iverson 1962), which converts a Boolean value to a number (false to 0, true to 1). However, in terms of Definition 6 above, we can see $=$ as a binary operator with $(f = g)(s) = 1$ if $f(s) = g(s)$, and $(f = g)(s) = 0$, otherwise. Thus, if only one package is loaded into the truck, driving to another location induces a cost of 2, while driving without any package induces a cost of 1. The support of a drive action consists therefore of the variables $\mathrm{p_1\text{-}at}$ and $\mathrm{p_2\text{-}at}$. To ease notation, we will stop mentioning the state in the cost function, and instead write $c_{\mathrm{drive}} = [\mathrm{p_1\text{-}at} = t] + [\mathrm{p_2\text{-}at} = t] + 1$, i.e. a variable in a cost function term refers to the state value of that variable.

We now have all definitions required to formalize a planning task:

Definition 7 (Planning task). A *planning task* is a tuple $\Pi = (\mathcal{V}, \mathcal{A}, s_I, s_\star, c)$ consisting of the following components:

- $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of finite-domain *state variables*,
- $\mathcal{A}$ is a set of *actions*,

- state $s_I \in \mathcal{S}$ is called the *initial state*,
- $s_\star$ is a partial state which specifies the *goal* condition,
- $c : \mathcal{A} \times \mathcal{S} \to \mathbb{Q}^+$ is the global cost function of Π.

A state s is a goal state iff $s_\star \subseteq s$. We denote the set of goal states by $S_\star$ and say variable v is a goal variable if $v \in \mathrm{vars}(s_\star)$. We call Π a *unit-cost task* if for all $a, a' \in \mathcal{A} : c_a(s) = c_{a'}(s')$ for all $s, s' \in \mathcal{S}$. We call it a *constant-cost task*, if for all $a \in \mathcal{A} : c_a(s) = c_a(s')$. Otherwise, we call Π a *state-dependent action cost task*, or simply planning task. If we talk about classical planning, we assume that planning tasks are either unit-cost or constant-cost tasks. In such cases, we will also simply write c_a instead of $c_a(s)$.

Example 1. For the sake of completeness, we give the complete definition of our logistics task:

- $\mathcal{V} = \{\text{t-at}, \text{p}_1\text{-at}, \text{p}_2\text{-at}\}$
- $\mathcal{D}_{\text{t-at}} = \{A, B, C\}$, $\mathcal{D}_{\text{p}_1\text{-at}} = \mathcal{D}_{\text{p}_2\text{-at}} = \{A, B, C, t\}$
- $\mathcal{A} = \{$

$$\begin{aligned}
\text{load-p}_1 &= \langle(\text{t-at} \doteq A) \wedge (\text{p}_1\text{-at} \doteq A), (\text{p}_1\text{-at} \doteq t)\rangle,\\
\text{load-p}_2 &= \langle(\text{t-at} \doteq B) \wedge (\text{p}_2\text{-at} \doteq B), (\text{p}_2\text{-at} \doteq t)\rangle,\\
\text{unload-p}_1 &= \langle(\text{t-at} \doteq C) \wedge (\text{p}_1\text{-at} \doteq t), (\text{p}_1\text{-at} \doteq C)\rangle,\\
\text{unload-p}_2 &= \langle(\text{t-at} \doteq C) \wedge (\text{p}_2\text{-at} \doteq t), (\text{p}_2\text{-at} \doteq C)\rangle,\\
\text{drive-AB} &= \langle(\text{t-at} \doteq A), (\text{t-at} \doteq B)\rangle,\\
\text{drive-BA} &= \langle(\text{t-at} \doteq B), (\text{t-at} \doteq A)\rangle,\\
\text{drive-AC} &= \langle(\text{t-at} \doteq A), (\text{t-at} \doteq C)\rangle,\\
\text{drive-CA} &= \langle(\text{t-at} \doteq C), (\text{t-at} \doteq A)\rangle,\\
\text{drive-BC} &= \langle(\text{t-at} \doteq B), (\text{t-at} \doteq C)\rangle,\\
\text{drive-CB} &= \langle(\text{t-at} \doteq C), (\text{t-at} \doteq B)\rangle\}
\end{aligned}$$

- $s_I = (\text{t-at} \doteq A) \wedge (\text{p}_1\text{-at} \doteq A) \wedge (\text{p}_2\text{-at} \doteq B)$
- $s_\star = \{(\text{p}_1\text{-at} \doteq C), (\text{p}_2\text{-at} \doteq C)\}$
- $c_{\text{load-p}_1} = c_{\text{load-p}_2} = c_{\text{unload-p}_1} = c_{\text{unload-p}_2} = 1$,
- $c_{\text{drive-AB}} = c_{\text{drive-BA}} = c_{\text{drive-AC}} = c_{\text{drive-CA}} = c_{\text{drive-BC}} = c_{\text{drive-CB}} =$

$$[\text{p}_1\text{-at} = t] + [\text{p}_2\text{-at} = t] + 1.$$

Throughout the thesis we make the following assumptions.

Assumption 1 (No trivially inapplicable actions). *We assume that there are no trivially inapplicable actions, i.e.* pre *does not contain facts* $(v \doteq d)$ *and* $(v \doteq d')$ *for some* $v \in \mathcal{V}$ *and values* $d \neq d'$.

Assumption 2 (No trivially unsolvable tasks). *We assume the planning task not to be trivially unsolvable, i.e.* $s_\star$ *does not contain two facts* $(v \doteq d)$ *and* $(v \doteq d')$ *for some* $v \in \mathcal{V}$ *and values* $d \neq d'$.

Assumption 3 (Precondition and action cost variables are disjoint). *We assume that for all actions, no variable mentioned in the action precondition is required to compute the action cost, i.e.* $\mathrm{vars}(c_a) \cap \mathrm{vars}(\mathrm{pre}(a)) = \emptyset$.

All these assumptions can be guaranteed by a simple preprocessing step. In the case of trivially inapplicable actions we just remove the actions from the task. In the case of trivially unsolvable tasks we can immediately return that the task is unsolvable. For the last assumption, assume there are variables in $\mathrm{vars}(c_a)$ which are also mentioned in the precondition of a. Let v be such a variable, i.e. $v \in \mathrm{vars}(\mathrm{pre}(a)) \cap \mathrm{vars}(c_a)$. In this case, the value of v is already determined by $\mathrm{pre}(a)$ in each state s where a is applicable: $s(v) = \mathrm{pre}(a)(v)$. We can therefore transform c_a and substitute v with $\mathrm{pre}(a)(v)$. The resulting function will be equivalent to c_a in all states where a is applicable. We require one more assumption about the Cartesian sets of states we consider in this thesis.

Assumption 4 (Cartesian domains are not empty). *Let* $s_C = D_1 \times \ldots \times D_n$ *be a Cartesian set of states. We assume* $D_i \neq \emptyset$ *for* $i \in \{1, \ldots, n\}$.

We now finally define what a solution to a planning task is.

Definition 8 (Plan). Let Π be a planning task and let $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ be a sequence of actions from $\mathcal{A}$. We call π *applicable* in state s_0 if there exist states $s_1, \ldots, s_n$ such that a_i is applicable in s_i and $s_{i+1} = s_i[a_i]$ for all $i = 0, \ldots, n-1$, and denote s_n with $s_0[\pi]$. We call π an s-plan for Π if it is applicable in s and if $s_n \in S_\star$. If $s = s_0$ then we also simply say π is a plan for Π. The *cost* of an s_0-plan π is the sum of action costs along the induced state sequence, i.e., $cost(\pi) = \sum_{i=0}^{n-1} c_{a_i}(s_i)$. An *optimal* s-plan is an s-plan that minimizes $cost(\pi)$.

In our example task, there are infinitely many plans to reach the goal, since we can drive endlessly around as long as we drop both packages at C at some point. However, there are only three optimal plans, i.e. plans with minimal cost. We can load p_1, drive to C and unload p_1, drive to B, load p_2, and drive back to C and unload the package. This requires two load and two unload actions with a total cost of 4, two drive actions with a single package loaded with a total cost of 4 and one drive action with no package loaded with a total cost of 1, i.e. the total plan cost is $4 + 4 + 1 = 9$. Another plan is to

load p_1, drive to B, load p_2 and drive to C and unload both packages. Again, we have two load and two unload actions with a total cost of 4. But this time one drive action loaded with a single package with cost 2 and one drive action loaded with both packages with a cost of 3, and therefore total plan cost $4+2+3=9$. The third plan is the same, but we swap the order in which we unload the packages. Note that if we would increase the unconditional cost of driving to 2 instead of only 1 we would end up with the latter two plans being the only optimal plans (with a cost of 11).

2.1.1 Heuristic Search

Finding a plan or even proving that one exists for a classical planning problem is already PSPACE-hard (Bäckström and Nebel 1995; Bylander 1994). Finding an optimal plan is still PSPACE-hard, but in practice usually much harder than finding any plan (Helmert 2003). As we require cost functions to be evaluatable in polynomial time, these results carry over to planning with state-dependent action costs.

Although it is unlikely that there exists a polynomial time planning algorithm, there are still powerful tools which allow to find plans for a variety of domains of different difficulty and size (Helmert 2006a; Torralba and Pommerening 2018). One of the most powerful tools to find a plan is *heuristic search*, where we reason over the *transition system* induced by the planning task.

Definition 9 (Transition system). A planning task Π induces a labeled, weighted *transition system* $\mathcal{T}_\Pi = (\mathcal{S}, L, T, s_I, S_\star)$ with state space $\mathcal{S}$, transition labels L, transition relation $T \subseteq \mathcal{S} \times L \times \mathbb{Q}^+ \times \mathcal{S}$, initial state s_I and goal states $S_\star$. Labels L correspond to the actions of Π, and for each action $a \in \mathcal{A}$ and state $s \in \mathcal{S}$ there is a transition $(s,\ a,\ w,\ s[a])$ with label a and weight $w = c_a(s)$ if and only if a is applicable in s. We write $s \overset{a,c_a(s)}{\longrightarrow} s[a]$ for $(s,\ a,\ c_a(s),\ s[a])$.

Example 2. Figure 2.3 depicts the transition system of the logistics task of Example 1. Each node represents the values for the variables in order $\mathrm{t\text{-}at}$, $\mathrm{p_1\text{-}at}$, $\mathrm{p_2\text{-}at}$. Drive action labels are omitted, and instead the cost of drive actions are denoted between states. The initial state is marked in grey, goal states are marked by double lined nodes. Two (of the three) cheapest cost plans are highlighted red and blue, respectively.

A cheapest path in $\mathcal{T}_\Pi$ from the initial state to some goal state is therefore a sequence of actions which corresponds to an optimal plan for Π. Thus, the problem of finding an optimal plan is equivalent to the problem of finding a cheapest path in the transition system from the initial state to some goal state. The $A^\star$ algorithm (Hart et al. 1968) is a best-first search algorithm and has been applied successfully in many different branches of AI, such as

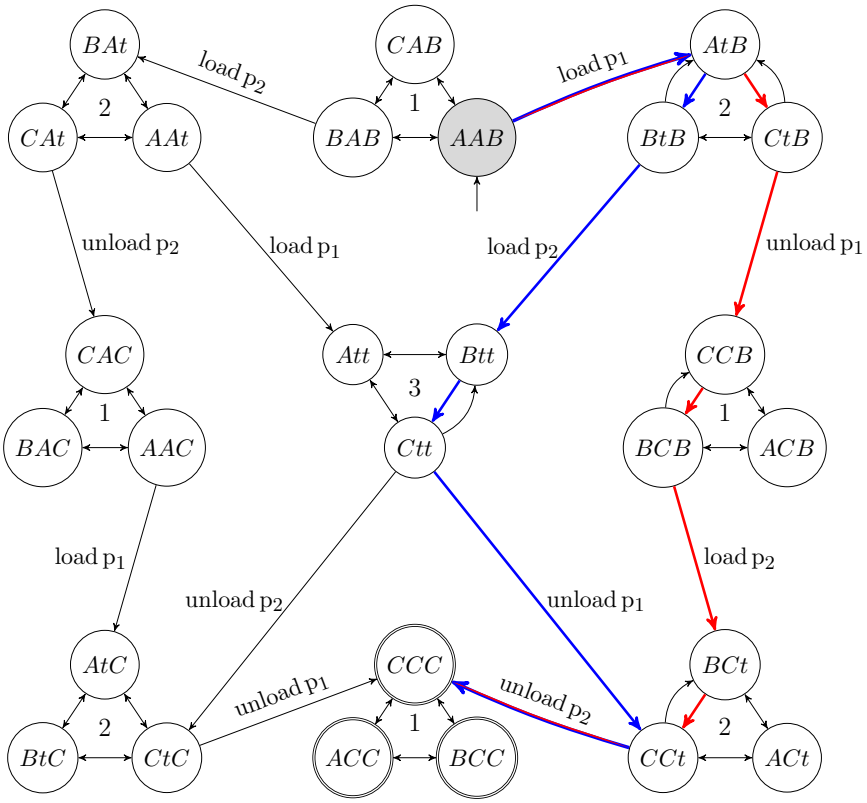


Figure 2.3: Transition system of the logistics task in Example 1. Each node depicts a state for variables $\mathrm{t\text{-}at}, \mathrm{p}_1\text{-}\mathrm{at}, \mathrm{p}_2\text{-}\mathrm{at}$. Labels for drive actions are omitted, instead the numbers between nodes represent the cost of the drive actions. Two plans are highlighted in red and blue. Both plans have a cost of 9.

route-planning (Bast et al. 2016), video games, or multi-agent planning (Silver 2005). In optimal AI planning, $A^\star$ is one of the most commonly used algorithms to find optimal plans. The algorithm estimates the minimum cost of reaching a goal from the initial state via a function $f(s) = g(s) + h_\Pi(s)$ and progressively "expands" states s with minimum f-value. Here, $g(s)$ is the cost of a current path from s_I to s. The heuristic h_Π estimates the distance from some state s of Π to a goal state. From now on, we will omit labeling the heuristic with the corresponding task and the task will be clear from the context. A heuristic is a function $h : \mathcal{S} \to \mathbb{N} \cup \{\infty\}$ and the perfect heuristic $h^\star$ maps each state $s \in \mathcal{S}$ to the cost of an optimal s-plan, or to ∞ if no such plan exists. In the following, we present some important concepts regarding heuristics.

Definition 10 (Heuristic properties). Let Π be a planning task and $\mathcal{T}_\Pi$ its transition system with state space $\mathcal{S}$. Let $h : \mathcal{S} \to \mathbb{N} \cup \{\infty\}$ be a heuristic. We

say that h is *goal-aware* if $h(s) = 0$ for $s \in S_\star$. We say that h is *admissible* if it never overestimates the true cost, i.e. if $h(s) \leq h^\star(s)$ for all $s \in \mathcal{S}$. Finally, h is *consistent*, if $h(s) \leq c_a(s) + h(s[a])$ for all $a \in \mathcal{A}$ and $s \in \mathcal{S}$.

Furthermore, given two heuristics h and h', we say that h *dominates* h' if and only if $h(s) \geq h'(s)$ for all $s \in \mathcal{S}$.

If an admissible heuristic is used, the $A^\star$ algorithm returns optimal plans. Heuristics which are goal-aware and consistent are admissible (Russel and Norvig 2010) and heuristics which are admissible are goal-aware. While there are exceptions (Holte 2010), search is in general more efficient if an admissible heuristic yields a high estimate, and the search for more powerful admissible heuristics has dominated the field of optimal classical planning for many years.

For tasks with state-dependent action costs, computing the g-value of a state is simple: the algorithm just has to evaluate the action cost function given a state s and add the g-value of the predecessor state. Heuristics, however, have been developed for tasks with constant action costs, and this brings up the question if and how we can generalize different heuristics to tasks with state-dependent action costs. But before we delve into such questions, we first discuss how we can express state-dependent action costs from the modeling perspective, and take a look at different representations of action cost functions.

2.2 Modeling State-dependent Action Costs

Most research in planning is concerned with classical planning, and all tasks of the previous International Planning Competitions (Muise 2015) only involve unit or constant action cost. While there are extensions to classical planning, e.g. numeric planning (Fox and Long 2003), or planning with constraints and preferences (Gerevini and Long 2005), none of these explicitly mention state-dependent action costs. In the following, we discuss how we can model state-dependent action cost tasks with current planning formalisms.

2.2.1 PDDL

The *planning domain definition language* (PDDL) (McDermott et al. 1998), originally introduced in 1998, is the prevalent language to model planning tasks and also the language used in the various International Planning Competitions (ICAPS competitions 2018). Therefore, there exists a large benchmark set which has grown over the years, and the language itself has received various extensions. The *plan metric* field, introduced with PDDL 2.1 (Fox and Long 2003) allows to specify, among others, that plan costs should be minimized. Action costs can then be expressed with an $(\text{INCREASE TOTAL-COST } k)$ effect, which denotes that applying the action has an effect of a cost increase

of k. *Conditional effects* allow actions to have effects which only trigger if some condition is met, independent of the unconditional action effect. This already allows us to model simple tasks with state-dependent action costs. For example, our logistics planning problem might model the drive actions with a conditional effect where the condition is that package p_1 is loaded in the truck, and the effect is $(\text{INCREASE TOTAL-COST } 1)$.

This allows us to arbitrarily sum up different conjunctions of facts, but we are still not able to efficiently express subtraction or multiplication in our cost functions. In principle, we can also use conditional effects to represent arbitrary state-dependent cost functions, simply by enumerating all possible partial states of the support of the action cost, and expressing each partial state as a condition of a conditional effect. However, such a task will be of exponential size in the number of variables in the support. We will come back to this later, when we define different compilations of state-dependent action cost tasks into classical planning tasks, in Section 2.4.

PDDL 2.1 also extended the language with the ability to specify numeric expressions, which is the foundation for the field of *planning with numerical state variables* (in short: numeric planning), where the domain of a variable may assume real numbers, and actions can have arbitrarily complex effects, including multiplication and subtraction. Ivankovic et al. (2014) already recognized that state-dependent action costs can be represented with unrestricted numeric PDDL. However, while classical planning is PSPACE-hard, numeric planning is undecidable (Helmert 2002), and various work is concerned with extending concepts from classical planning to numeric planning (Aldinger and Nebel 2017; Eyerich et al. 2009; Scala et al. 2017). As such, while we can express complex cost functions with numeric PDDL, it is out of scope for our research in classical planning with state-dependent action costs.

With PDDL 3.0 (Gerevini and Long 2005), *plan constraints and preferences* were introduced. These also allow to model state-dependent action costs, as "preferences in action preconditions may incur a penalty when applying the action in a state where the preference is unsatisfied" (Ivankovic et al. 2014). Ceriani and Gerevini (2015) make use of this relation and present a compilation of *always preferences*, conditions which should hold in every state visited by application of a plan, to a classical planning problem with action costs. The PDDL 3.0 planner HPLAN-P (Baier et al. 2007) compiles preferences in action preconditions away by introducing a counter which tracks the number of preference violations. In the plan metric, this counter is used to increase the cost of the overall plan. In principle, this can be seen as another form of achieving state-dependent action costs by making use of conditional effects together with plan metric properties.

2.2.2 Fast Downward Input Language

While extensions to PDDL allow to express state-dependent action costs, all extensions incur a strong addition to the planning formalism itself. PDDL 2.1 allows numeric planning, and PDDL 3.0 allows in principle to express arbitrary formulae in linear temporal logic. As a consequence, any planner supporting one of these fragments of PDDL has to be able to solve a superset of state-dependent action cost tasks. However, in this thesis, we are more concerned with understanding state-dependent action costs, and therefore examine this topic in the purest possible setting, i.e. classical planning extended with state-dependent action costs. While a planner-independent formalism to express classical planning tasks with state-dependent action costs would be preferable, proposing yet another planning formalism is out of the scope of this thesis, and requires cooperation among the planning community. We therefore mention another practical way to express state-dependent action costs, which is more concerned with the planner input language itself, as opposed to an expressive language usable by different planning engines.

The Fast Downward planning system (Helmert 2006b) is a state-of-the-art planning system which is used and extended by many researchers around the world. In the optimal planning track of the recent International Planning Competition (Torralba and Pommerening 2018), 11 out of 12 planners (this excludes multiple versions of the same planning system) are based or make heavy use of Fast Downward. In principle, Fast Downward consists of two components: a translator component takes a PDDL domain and problem file, applies various transformations (Helmert 2009) (e.g. invariant synthesis, reachability analysis) and returns a file used by the search component. This internal file represents an SAS$^+$ planning task formulation, and as such also contains fields that express variables and domains, as well as actions and their cost. Later in the thesis, we will evaluate different approaches to state-dependent action cost planning, and their implementation is based on Fast Downward. Therefore, we express our planning tasks in the input language format of Fast Downward. To allow state-dependent action cost we only have to modify the action cost field in this specification.

2.3 Representing State-dependent Action Costs

Expressing state-dependent action costs is important for modeling planning problems, but it does not answer the question of how to algorithmically deal with state-dependent costs. When we are not interested in heuristics and apply a simple forward search algorithm (e.g. $A^\star$ with $h(s) = 0$ for all states s) it suffices to be able to compute the action cost for a given state, e.g. representing the action cost function as a C-like programming function. However, we will see that it may be beneficial to view the action cost function on a more "symbolic" level, and exhibit and exploit structure hidden in the cost function.

To achieve this, we will use decision diagrams as a representation of our cost functions. Decision diagrams play an important role in formal verification of digital circuits and communication protocols (Clarke et al. 1999) and have also been applied to other areas of AI, such as constraint logic programming (Codognet and Diaz 1996), stochastic planning (Hoey et al. 1999), computing shortest paths (Bahar et al. 1997), discrete optimization (Bergman et al. 2016), and heuristic search (Edelkamp and Reffel 1998; Hansen et al. 2002). In the past decade, decision diagrams have also been successfully applied to AI planning (Edelkamp and Helmert 2001; Edelkamp and Kissmann 2009; Torralba 2015). Here, decision diagrams are used to symbolically represent sets of states and action transitions, and a "symbolic planner" performs search on sets of states instead of doing explicit state-space search. In 2014, the symbolic planner SYMBA* was the winner of the International Planning Competition (Edelkamp et al. 2015). While we are not concerned with the question of how to plan with symbolic representations, we will apply symbolic representations of cost functions and reason over these representations. Therefore, we give a brief introduction to several types of decision diagrams, and focus on so called edge-valued multi-valued decision diagrams later on.

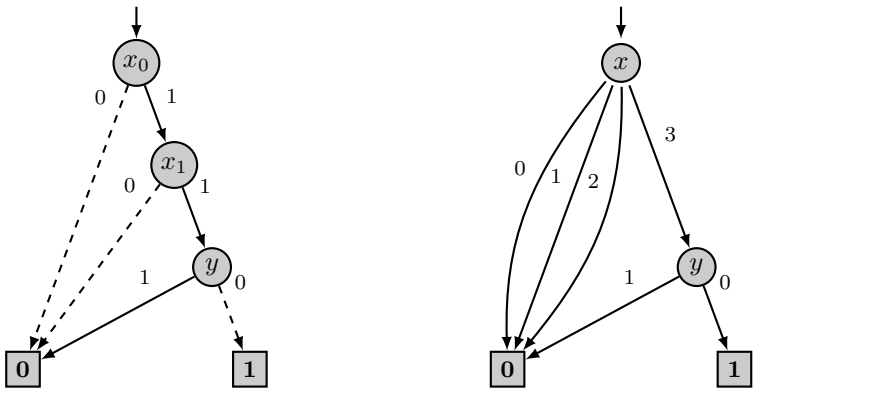
2.3.1 Binary Decision Diagrams

The most prominent representative of decision diagrams, and also the predecessor for many other diagrams, are so called *Binary Decision Diagrams* (BDDs) (Akers 1978; Bryant 1986; Lee 1959). They are based on *Boole's expansion theorem* (Boole 1854), which is also often called Shannon expansion.

Theorem 1 (Boole's expansion theorem). *Let $f : \{0,1\}^n \to \{0,1\}$ be a function with $n \in \mathbb{N}$ and let $f|_{x_i=1}$ be the function where x_i is substituted with 1 in f (analogously with 0) for $i \in \{1,\dots,n\}$. Then $f(x_1,\dots,x_n) = \neg x_i \cdot f|_{x_i=0} + x_i \cdot f|_{x_i=1}$.*

A BDD over binary variables $\mathcal{V} = \{v_1,\dots,v_n\}$ is a directed acyclic graph and represents a Boolean function $f : \{0,1\}^n \to \{0,1\}$. A BDD has a single root node and two terminal nodes **0** and **1**, also called the 0-sink and the 1-sink. A non-terminal node **v** is associated with variable $v \in \mathcal{V}$ and denotes the Boolean function $(\neg v \wedge \chi_{low}) \vee (v \wedge \chi_{high})$, where χ_{low} and χ_{high} are successors of **v** corresponding to the functions where v is false (χ_{low}), true (χ_{high}), respectively. Evaluation of f then equates to the traversal of the BDD starting at the root node and traversing edges corresponding to the assignment of the variables until a terminal node is reached. If the 0-sink is reached, the value of f is 0, otherwise it is 1. Figure 2.4a depicts a BDD for function $f = x_0 \wedge x_1 \wedge \neg y$.

BDDs have several properties which make them preferable for representation and transformation of Boolean functions (Bryant 1986). First, for many commonly-encountered functions, the size of the BDD is relatively small. Second, the complexity of any operation on two functions is bounded by the



(a) BDD representing $f = x_0 \wedge x_1 \wedge \neg y$ (b) MDD representing $f = (x \doteq 3) \wedge (y \doteq 0)$

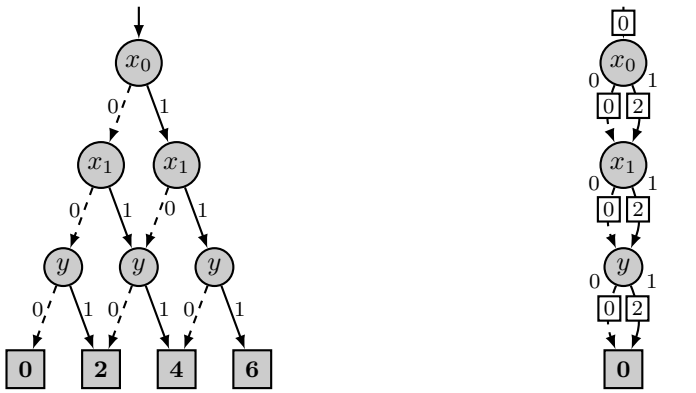
Figure 2.4

product of the BDD sizes for these functions. Finally, by imposing restrictions on the ordering of the decision variables in the BDD and by eliminating redundant vertices, a BDD is a canonical form of a function, i.e. every function has a unique BDD representation. Such BDDs are sometimes called *reduced ordered binary decision diagrams* (ROBDDs).

In principle, BDDs can also be applied to problems where variables have a non-binary domain. The key idea is to encode each variable v_i with $\lceil \log_2 |\mathcal{D}_{v_i}| \rceil$ additional variables. However, we can generalize the definition of BDDs to *multi-valued decision diagrams* (MDDs), which naturally allow for variables with non-binary domain by having more than two outgoing edges for each node (Miller 1993; Srinivasan et al. 1990). Figure 2.4b depicts an MDD for function $f = (x \doteq 3) \wedge (y \doteq 0)$, i.e. the function maps only state s with $s(x) = 3$ and $s(y) = 0$ to 1.

2.3.2 Multi-terminal Decision Diagrams

Another way to generalize BDDs is to seek for a representation of functions with non-Boolean codomain, i.e. the function maps to other sets than to $\{0, 1\}$. For this, Clarke et al. (1993a,b) generalize BDDs to have multiple (more than two) terminal nodes, which can then represent arbitrary integers. They call this type of diagrams *multi-terminal binary decision diagrams* (MTBDDs) and allow the values of terminal nodes to be of arbitrary sets, but they primarily represent matrices in their work. Bahar et al. (1993, 1997) extend this concept: they develop a theory of MTBDDs which represent functions $f : \{0, 1\}^n \to M$, where M is the carrier of some algebraic structure and the values of terminal nodes are therefore of type M. While these diagrams are still MTBDDs, they use the term *algebraic decision diagram* (ADD) instead, to emphasize their use for arbitrary algebraic structures, and they evaluate them



(a) ADD (b) EVBDD

Figure 2.5: ADD and EVBDD representation of $f = 2x_0 + 2x_1 + 2y$.

on a number of different applications, including shortest-path problems and solving systems of linear equations. Figure 2.5a depicts an ADD for function $f = 2x_0 + 2x_1 + 2y$, where the carrier set is $\mathbb{N}$.

2.3.3 Edge-valued Decision Diagrams

So far we have seen two ways of extending BDDs: allowing for more than two terminal nodes (ADDs/MTBDDs), and allowing for non-terminal nodes with more than two children (MDDs). Another form of decision diagram changes the way the diagram is evaluated by extending edges with weights. Lai and Vrudhula[1] (1992) introduced *edge-valued* binary decision diagrams (EVBDDs), which are more suited if one seeks to represent arithmetic, instead of Boolean functions. They apply them to several applications, such as integer linear programming, spectral transformations, and multiple-output decomposition of Boolean functions. EVBDDs have three important distinctions to BDDs: the root node has a dangling incoming edge, there is only a single terminal node **0**, and edges are weighted. Instead of denoting a Boolean function, a node **v** in the EVBDD associated with (binary) variable v denotes the arithmetic function $(1 - v)(w_{low} + \chi_{low}) + v \cdot (w_{high} + \chi_{high})$, where w_{low} and w_{high} are weights on the edges to the children χ_{low} and χ_{high}. Again, these children represent arithmetic functions themselves. While for BDDs, the evaluation of function f depends on the terminal node, for EVBDDs, the evaluation of function f equates to the sum of the edge weights along the path corresponding to the assignment of the variables. Figure 2.5b depicts an EVBDD for function $f = 2x_0 + 2x_1 + 2y$.

If the value of w_{low} is fixed to 0, then reduced and ordered EVBDDs also have the canonical property. Additionally, EVBDDs can be exponentially

[1] S. Vrudhula published this work as S. Sastri.

more compact than ADDs (Lai et al. 1996). Take for example the function $f = \sum_{i=1}^{n} 2^i \cdot v_i$. The EVBDD representation of f requires $n + 1$ nodes, since the weights can encode the partial values of f corresponding to the value of variable v_i. The ADD representation of f, however, requires $2^{n+1} - 1$ nodes, as the terminal nodes have to represent all possible values that f can take on.

While EVBDDs never require more nodes than ADDs, there is an important difference between the time complexity of operations performed on them. For BDDs and ADDs the complexity is bounded by the product of the BDD (ADD) sizes; this is not necessarily the case for EVBDDs. We won't describe the details here (and refer to Lai et al. (1996) instead), but note that if the EVBDD operations satisfy some specific properties (called the additive property and the bounding property), then time complexity is greatly reduced. Additionally, EVBDDs can also be used to represent Boolean functions. In this case they have the same size and require the same time complexity for performing operations. We should mention, however, that in practice the edge-weighted nature of EVBDDs makes them inferior to BDDs for Boolean function representation, due to the additional overhead of representing edge weights.

2.3.4 Edge-valued Multi-valued Decision Diagrams

The generalization of EVBDDs to multi-valued variables is called *Edge-valued Multi-valued Decision Diagram* (EVMDD) (Ciardo and Siminiceanu 2002) (often also denoted as EV$^+$MDD) and will be the primary data structure we will use to represent cost functions in planning tasks with state-dependent action costs. In general, EVMDDs may also be defined over other algebraic structures (Mattmüller et al. 2018), but we restrict our definitions to the case where an EVMDD represents a cost function $c : \mathcal{S} \to \mathbb{Q}^+$. Therefore, we will formally introduce EVMDDs and also formally define reduced and ordered EVMDDs, and note that this is analogous to the requirements on reduced and ordered BDDs (ADDs, EVBDDs). Obviously, the aforementioned comparison of EVBDD and ADD sizes also holds for EVMDDs.

Definition 11 (Edge-valued multi-valued decision diagram). An *EVMDD* over a finite set of variables $\mathcal{V}$ is a tuple $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$, where $\kappa \in \mathbb{Q}^+$ is a constant value and $\mathbf{f}$ is a directed acyclic graph consisting of two types of nodes: (1.) There is a single *terminal node* denoted by $\mathbf{0}$. (2.) A *nonterminal node* $\mathbf{v}$ is a tuple $(v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ where $v \in \mathcal{V}$ is a variable, $k = |\mathcal{D}_v| - 1$, the children $\chi_0, \ldots, \chi_k$ are terminal or nonterminal nodes of $\mathcal{E}$, and $w_0, \ldots, w_k \in \mathbb{Q}^+$ s.t. $\min_{i=0,\ldots,k} w_i = 0$ are the weights assigned to the edges to the children.

By κ we refer to the input weight of $\mathcal{E}$, and by $\mathbf{f}$ we also refer to the root node of $\mathcal{E}$. We refer to the components of $\mathbf{v}$ as $\mathrm{var}(\mathbf{v}), \chi_i(\mathbf{v})$ and $w_i(\mathbf{v})$. Edges of $\mathcal{E}$ between parent and child nodes are implicit in the definition of the nonterminal nodes of $\mathcal{E}$. The *weight* of an edge from $\mathbf{v}$ to $\chi_i(\mathbf{v})$ is $w_i(\mathbf{v})$. We

denote the number of nodes in $\mathcal{E}$ with $|\mathcal{E}|$. The following definition specifies the arithmetic function denoted by a given EVMDD.

Definition 12 (Arithmetic function denoted by the EVMDD). An EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ denotes the arithmetic function $\kappa + f$ where f is the function denoted by $\mathbf{f}$. The terminal node $\mathbf{0}$ denotes the constant function 0, and $(v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ denotes the arithmetic function over $\mathcal{S}$ given by $f(s) = f_{s(v)}(s) + w_{s(v)}$, where $f_{s(v)}$ is the arithmetic function denoted by child $\chi_{s(v)}$. We write $\mathcal{E}(s)$ for $\kappa + f(s)$.

This means that, given an EVMDD $\mathcal{E}_f$ encoding function $f : \mathcal{S} \to \mathbb{Q}^+$, and given a state s, the value $f(s)$ can be read from $\mathcal{E}_f$ as the sum of edge weights along the unique EVMDD path corresponding to s, $\mathcal{E}_f(s)$.

In the graphical representation of an EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$, $\mathbf{f}$ is represented by a rooted directed acyclic graph and κ by a dangling incoming edge to the root node of $\mathbf{f}$. The terminal node is depicted by a rectangular node labeled $\mathbf{0}$. Edge labels d are written next to the edges, edge weights w_d in boxes on the edges.

Example 3. Consider the cost function of the drive actions of the logistics task presented in Example 1: $c_{\text{drive-AB}} = [\text{p}_1\text{-at} = t] + [\text{p}_2\text{-at} = t] + 1$. Figure 2.6 depicts the EVMDD representation of $c_{\text{drive-AB}}$. Evaluating state $s = (\text{t-at} \doteq A) \wedge (\text{p}_1\text{-at} \doteq t) \wedge (\text{p}_2\text{-at} \doteq t)$ corresponds to the sum of the edge weights along the path $(\text{p}_1\text{-at} \doteq t)$ and $(\text{p}_2\text{-at} \doteq t)$, resulting in a cost of 3.

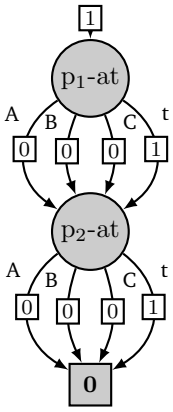


Figure 2.6: EVMDD of the drive cost function $[\text{p}_1\text{-at} = t] + [\text{p}_2\text{-at} = t] + 1$.

The EVMDD in Example 3 is already *ordered*. Informally speaking, an EVMDD is ordered if we can assign an ordering on the variables, and nodes in the EVMDD respect that ordering, i.e. the variable corresponding to a node in

the EVMDD has a higher order than the variable corresponding to one of the child nodes.

Definition 13 (Ordered EVMDD). An EVMDD $\mathcal{E}$ is *ordered* if there exists an ordering $level : \mathcal{V} \to \{1, \ldots, |\mathcal{V}|\}$ on the variables such that for every nonterminal node $\mathbf{v}$, either $\chi_i(\mathbf{v})$ is a terminal node, or we have $level(\mathrm{var}(\mathbf{v})) > level(\mathrm{var}(\chi_i(\mathbf{v})))$, for $i \in \{0, \ldots, |\mathcal{D}_{\mathrm{var}(\mathbf{v})}|-1\}$. If $\mathbf{v}$ is the terminal node, then $level(\mathbf{v}) = 0$.

Given an EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$, the order of $\mathcal{E}$ is $level(\mathrm{var}(\mathbf{f}))$. We will sometimes denote the ordering as a tuple $(v_n, \ldots, v_1)$ which means $level(v_i) = i, i \in \{1, \ldots, n\}$. This can be understood as "v_n appears above v_{n-1}" in the graphical representation.

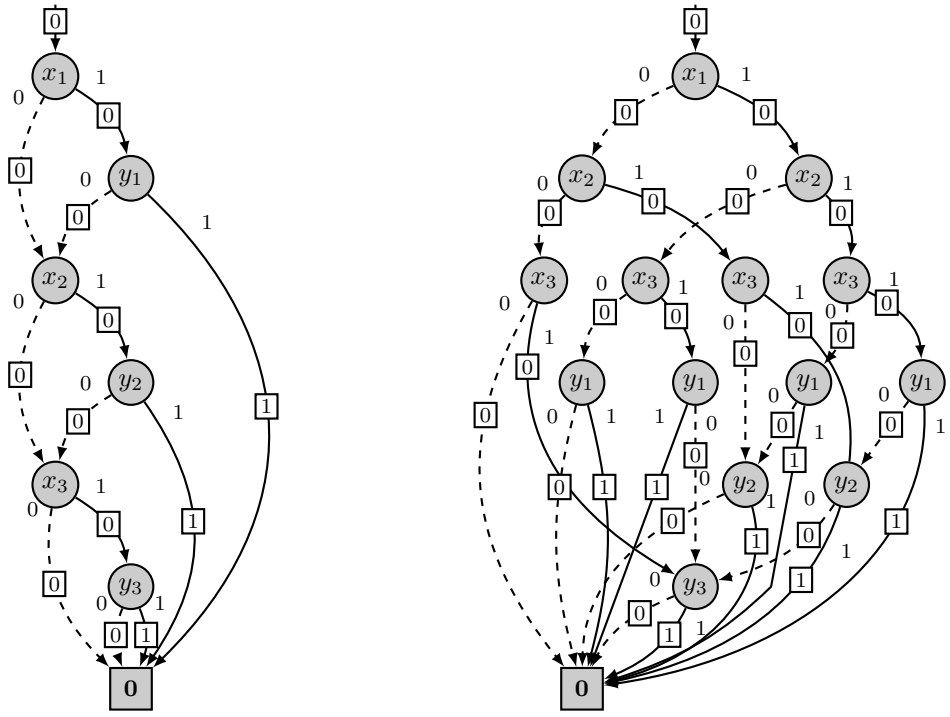
For EVMDDs, we will define two different properties of reducedness. The *reduced* property for EVMDDs corresponds to the same property for BDDs, but we will also define the *quasi-reduced* property, which guarantees that every path from the root node to the terminal node involves all variables.

Definition 14 (Reduced EVMDD). An ordered EVMDD is *reduced* if there is no nonterminal node $\mathbf{v} = (v, \chi_1, \ldots, \chi_k, 0, \ldots, 0)$ with $\chi_i = \chi_j$ for all $i, j \in \{1, \ldots, k\}$. [2]

Definition 15 (Quasi-reduced EVMDD, Ciardo and Siminiceanu 2002). An ordered EVMDD is *quasi-reduced* if there are no two nonterminal nodes $\mathbf{u}, \mathbf{v}$ such that $\mathbf{u} = \mathbf{v}$, and all edges span exactly one level, i.e. $level(\mathrm{var}(\chi_i)) = level(v) - 1$, $i \in \{1, \ldots, k\}$ for all nonterminal nodes $\mathbf{v} = (v, \chi_1, \ldots, \chi_k, w_1, \ldots, w_k)$.

Reduced and ordered EVMDDs are sometimes called ROEVMDDs. From now on we assume that all EVMDDs are either reduced or quasi-reduced according to some given ordering. As we do not care about uniqueness, we may use different, appropriate variable orderings for different cost functions. This may be beneficial, since different orderings have different effects on the size of an EVMDD. In particular, a good ordering can result in an exponentially more compact EVMDD. Take for example (Edelkamp and Kissmann 2011; Speck 2018) the two EVMDDs in Figure 2.7, where both diagrams depict the same function, but the size of one is polynomial in the number of variables (2.7a), while the other is exponential (2.7b). This raises the question if we can come up with an *optimal* ordering. Unfortunately, computing an optimal ordering is already co-NP-complete for BDDs (Bryant 1986). Given a BDD for a function f, it is even NP-complete to decide if there exists a BDD for f with some

[2] In the literature, there is also the additional requirement that there are no two different nonterminal nodes $\mathbf{u}, \mathbf{v}$ such that $\mathrm{var}(\mathbf{u}) = \mathrm{var}(\mathbf{v})$ and for all $i \in \{1, \ldots, k\}$ $\chi_i(\mathbf{u}) = \chi_i(\mathbf{v})$ and $w_i(\mathbf{u}) = w_i(\mathbf{v})$. As nodes, according to our definition, do not have a name (or identity), this implies $\mathbf{u} = \mathbf{v}$, therefore this requirement is redundant.



(a) variable order $x_1, y_1, x_2, y_2, x_3, y_3$. (b) variable order $x_1, x_2, x_3, y_1, y_2, y_3$.

Figure 2.7: Two EVMDDs with different variable orderings for function $f = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$.

size bound b (Bollig and Wegener 1996). Additionally, for some BDDs the size of the diagram is exponential, independent of the underlying ordering. Edelkamp and Kissmann (2011) show this for a BDD which encodes the goal condition of the game Connect Four. Their result carries over to the EVMDD case, as the representation is nearly identical (instead of edges in the BDD connecting to the **1** node, edges in the EVMDD have a weight of 1 and connect to the terminal node).

Nevertheless, for many functions, EVMDDs are quite compact and there exist techniques to come up with decent variable orderings. The *static ordering* approach determines an ordering prior to construction, and this order is maintained throughout all decision diagram operations. Most static orderings are based on heuristics. A comprehensive overview on some possible heuristics for BDDs and MDDs which also work for EVMDDs can be found in a survey by Rice and Kulhari (2008). *Dynamic ordering* (Rudell 1993) on the other hand performs reordering of variables in-between operations on the diagram. Such algorithms apply shifting of groups of symmetric variables (Panda and Somenzi 1995), simulated annealing (Bollig et al. 1995), or techniques

based on genetic algorithms (Drechsler et al. 1995). However, we note that since we are only interested in the representation of cost functions, and not in subsequent operations performed on this representation, dynamic orderings are not important for this work.

Sometimes we are required to speak more formally about paths in an EVMDD. If the EVMDD is quasi-reduced, then every path in the EVMDD corresponds to a unique valuation of the cost function.

Definition 16 (Path). Let $\mathcal{E}$ be an EVMDD with nodes $\mathbf{v}_0, \ldots, \mathbf{v}_n$. A sequence $p = \langle \mathbf{v}_n, d_{v_n}, \ldots, \mathbf{v}_1, d_{v_1}, \mathbf{v}_0 \rangle$ is a *path* from $\mathbf{v}_n$ to $\mathbf{v}_0$ if for $i \in \{n, \ldots, 1\}$ and $\mathbf{v}_i = \langle v_i, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k \rangle$ we have $\mathbf{v}_{i-1} = \chi_{d_{v_i}}$. Given a path p, we denote with $cost(p)$ the sum of weights along the path and with s_p the partial state corresponding to this path, i.e. $s_p = (v_n \doteq d_{v_n}) \wedge \ldots \wedge (v_1 \doteq d_{v_1})$.

Finally, we give the proof that reduced and ordered EVMDDs are a canonic representation, i.e. there exists a unique (up to isomorphism) representation of each function $f : \mathcal{S} \to \mathbb{Q}^+$. The proof is adapted from Ciardo and Siminiceanu (2002); their proof is for codomain $\mathbb{N} \cup \{\infty\}$ and only covers one direction, but the adaptation is quite straightforward.

Definition 17 (EVMDD isomorphism). Let $\mathcal{E}_f = \langle \kappa_f, \mathbf{f} \rangle$ and $\mathcal{E}_g = \langle \kappa_g, \mathbf{g} \rangle$ be two EVMDDs. Then $\mathbf{f}$ and $\mathbf{g}$ are *isomorphic*, if there exists a bijective function σ from the nodes of $\mathbf{f}$ to the nodes of $\mathbf{g}$ such that for any node $\mathbf{v}$ in $\mathbf{f}$, if $\sigma(\mathbf{v}) = \mathbf{v}'$, then either both $\mathbf{v}$ and $\mathbf{v}'$ are terminal nodes, or both are non-terminal nodes with $\mathrm{var}(\mathbf{v}) = \mathrm{var}(\mathbf{v}')$, $\sigma(\chi_i(\mathbf{v})) = \chi_i(\mathbf{v}')$ and $w_i(\mathbf{v}) = w_i(\mathbf{v}')$ for all $i \in \{0, \ldots, |\mathcal{D}_{\mathrm{var}(\mathbf{v})}| - 1\}$. If $\kappa_f = \kappa_g$, then $\mathcal{E}_f$ and $\mathcal{E}_g$ are *isomorphic*.

Lemma 1. *Let $level$ be some variable ordering, and $\mathbf{f}$ and $\mathbf{g}$ be two reduced and ordered (according to $level$) EVMDD nodes. Let $f : \mathcal{S} \to \mathbb{Q}^+$ and $g : \mathcal{S} \to \mathbb{Q}^+$ be the functions denoted by $\mathbf{f}$ and $\mathbf{g}$, respectively. If $\mathrm{var}(\mathbf{f}) \neq \mathrm{var}(\mathbf{g})$ then $f \neq g$.*

Proof. Since $\mathbf{f}$ and $\mathbf{g}$ are nodes of ordered and reduced EVMDDs, if we have $\mathrm{var}(\mathbf{f}) \neq \mathrm{var}(\mathbf{g})$, then either $level(\mathrm{var}(\mathbf{f})) > level(\mathrm{var}(\mathbf{g}))$ or $level(\mathrm{var}(\mathbf{f})) < level(\mathrm{var}(\mathbf{g}))$. Let without loss of generality $level(\mathrm{var}(\mathbf{f})) > level(\mathrm{var}(\mathbf{g}))$. Since $\mathbf{f}$ is reduced, there are states s and s' with $s(\mathrm{var}(\mathbf{f})) = d$, $s'(\mathrm{var}(\mathbf{f})) = d'$ such that $f(s) \neq f(s')$. However, such a state does not exist for g. Assume such a state exists, then there would have to be some successor node $\mathbf{v}$ of $\mathbf{g}$ with $\mathrm{var}(\mathbf{v}) = \mathrm{var}(\mathbf{f})$. Since $\mathbf{f}$ and $\mathbf{g}$ underlie the same ordering, and since $level(\mathrm{var}(\mathbf{f})) > level(\mathrm{var}(\mathbf{g}))$, this is a contradiction. □

Theorem 2 (EVMDD canonicity). *Given some variable ordering $level$, let $\mathcal{E}_f = \langle \kappa_f, \mathbf{f} \rangle$ be a reduced ordered EVMDD encoding function $f : \mathcal{S} \to \mathbb{Q}^+$ and let $\mathcal{E}_g = \langle \kappa_g, \mathbf{g} \rangle$ be a reduced ordered EVMDD encoding function $g : \mathcal{S} \to \mathbb{Q}^+$. Then, $f = g$ if and only if $\mathcal{E}_f$ and $\mathcal{E}_g$ are isomorphic.*

We first show that two isomorphic nodes encode the same function:

Lemma 2. *Let $level$ be some variable ordering, and* $\mathbf{f}$ *and* $\mathbf{g}$ *be two reduced and ordered (according to $level$) EVMDD nodes. Let $f : \mathcal{S} \to \mathbb{Q}^+$ and $g : \mathcal{S} \to \mathbb{Q}^+$ be the functions denoted by* $\mathbf{f}$ *and* $\mathbf{g}$*, respectively. Then $f = g$, if and only if* $\mathbf{f}$ *and* $\mathbf{g}$ *are isomorphic.*

Proof. By induction over the level of $\mathbf{f}$.

Base case $level(\mathbf{f}) = 0$: Both nodes $\mathbf{f}$ and $\mathbf{g}$ are terminal nodes and encode the function $f = 0 = g$, and are, by definition, isomorphic.

Inductive step $level(\mathbf{f}) = l > 0$: Assume the claim is true for all nodes $\mathbf{v}$ with $level(\mathbf{v}) \leq l - 1$. Let $i \in \{1, \ldots, k\}$ and $\mathcal{S}_i$ be the set of states s with $s(\text{var}(\mathbf{f})) = i$. We write χ_f for the function denoted by $\chi_i(\mathbf{f})$, w_f for $w_i(\mathbf{f})$, χ_g for the function denoted by $\chi_i(\mathbf{g})$, and w_g for $w_i(\mathbf{g})$.

First, assume that $f = g$. Then, $\chi_f(s) + w_f = \chi_g(s) + w_g$ for $s \in \mathcal{S}_i$, since $\chi_f + w_f$ denotes f and $\chi_g + w_g$ denotes g, respectively. Since there is always a zero-weighted path from a node to the terminal node, we have $\min_{s \in \mathcal{S}_i} \chi_f(s) = 0 = \min_{s \in \mathcal{S}_i} \chi_g(s)$, and therefore the minimum values that $\chi_f(s) + w_f$ and $\chi_g(s) + w_g$ can have are w_f and w_g, respectively. Then, $w_f = \min_{s \in \mathcal{S}_i} \chi_f(s) + w_f = \min_{s \in \mathcal{S}_i} \chi_g(s) + w_g = w_g$. This implies that $\chi_f = \chi_g$ and by induction hypothesis that $\chi_i(\mathbf{f})$ and $\chi_i(\mathbf{g})$ are isomorphic. Since i is arbitrary, and by contraposition of Lemma 1, it follows that $\mathbf{f}$ and $\mathbf{g}$ isomorphic.

Now, assume that $\mathbf{f}$ and $\mathbf{g}$ are isomorphic, then $w_f = w_g$. Since $\chi_i(\mathbf{f})$ and $\chi_i(\mathbf{g})$ are isomorphic, we get by induction hypothesis that $\chi_f = \chi_g$, and thus $\chi_f + w_f = \chi_g + w_g$. Again, since i is arbitrary, it follows that $f = g$. □

This brings us to the simple proof for Theorem 2.

Proof. Assume that $f = g$. Since all nodes have at least one outgoing edge with weight 0, there exists a state s with $\kappa_f = f(s) = g(s) = \kappa_g$. Therefore $\mathbf{f}$ and $\mathbf{g}$ must encode the same function $f - \kappa_f$ and it follows from Lemma 2 that they are isomorphic.

Now assume $\kappa_f = \kappa_g$, and $\mathbf{f}$ and $\mathbf{g}$ are isomorphic. $\mathcal{E}_f$ encodes function $f = \kappa_f + f'$, where f' is the function encoded by $\mathbf{f}$. Analogously, $\mathcal{E}_g$ encodes function $g = \kappa_g + g'$. From Lemma 2 it follows that $\mathbf{f}$ and $\mathbf{g}$ encode the same function, therefore $f' = g'$. Since $\kappa_f = \kappa_g$, we have $g = \kappa_g + g' = \kappa_f + f' = f$. □

2.3.5 Construction

We now discuss how to construct an EVMDD for a cost function $c : \mathcal{S} \to \mathbb{Q}^+$, given a fixed variable order $level$ and a fixed state space $\mathcal{S}$. According to Definition 6, we have to distinguish three cases:

1. c represents a constant function $c_a(s) = q$, for $q \in \mathbb{Q}^+, s \in \mathcal{S}$,
2. c represents a function $c(s) = s(v)$, for $s \in \mathcal{S}, v \in \mathcal{V}$,

3. c is the result of an operator $\circ : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ applied on two functions $f, g : \mathcal{S} \to \mathbb{Q}$.

Construction for the first case is straightforward: the corresponding EVMDD $\mathcal{E}_c$ for $c = q$ with $q \in \mathbb{Q}^+$ is $\mathcal{E}_c = \langle q, \mathbf{0} \rangle$, i.e. an EVMDD with input value q and the terminal node as its root node. For the second case, $c = v$ for some $v \in \mathcal{V}$, the corresponding EVMDD is $\mathcal{E}_c = (v, \mathbf{0}, \ldots, \mathbf{0}, 0, \ldots, |\mathcal{D}_v|-1)$, i.e. the weight of each edge is the corresponding domain value of that edge, and all edges lead to the terminal node. Observe that if $s(v) = d$, then $\mathcal{E}_c(s) = d$.

Construction for the last case is more involved. A central operation for this construction is the *apply* procedure (Lai et al. 1996). Let $\circ : \mathbb{Q}^+ \times \mathbb{Q}^+ \to \mathbb{Q}^+$ be an operator and $f : \mathcal{S} \to \mathbb{Q}^+$ and $g : \mathcal{S} \to \mathbb{Q}^+$ be two functions. We can, by slight abuse of notation, view $\circ$ as an operator on f and g with $(f \circ g)(s) = f(s) \circ g(s)$. Now, let $\mathcal{E}_{(\cdot)}$ be the construction that turns a function f into the reduced ordered EVMDD $\mathcal{E}_f$ representing f. We would like to have an operator $\circ_{\mathcal{E}}$ that mimics the behaviour of $\circ$ on EVMDDs, i.e. such that $\mathcal{E}_{f \circ g} = \mathcal{E}_f \circ_{\mathcal{E}} \mathcal{E}_g$. The *apply* procedure does exactly that. In the literature, the application of $\circ$ on the EVMDD level, $\mathcal{E}_f \circ_{\mathcal{E}} \mathcal{E}_g$, is usually written as $apply(\circ, \mathcal{E}_f, \mathcal{E}_g)$.

Algorithmically, the *apply* procedure traverses both input EVMDDs $\mathcal{E}_f$ and $\mathcal{E}_g$ from top to bottom in a synchronized manner, propagating edge weights downward, recursively applying $\circ$ to pairs of corresponding subgraphs with the same edge constraint, and pulling up excess edge weights again when the recursive computation has terminated. In the base case, when both EVMDDs represent constant functions encoded in their bottom-most edge labels w_f (in $\mathcal{E}_f$) and w_g (in $\mathcal{E}_g$), those get combined into the new edge label $w_f \circ w_g$.

Algorithm 1 describes a basic version of the $apply$ procedure. In line 2, function $terminal\_case$ checks if the base case applies, i.e. if both EVMDDs represent constant functions. Then, $apply$ returns an EVMDD with the combined constants (via $\circ$) as new input weight, and the terminal node $\mathbf{0}$ as root node. Otherwise, a new EVMDD is generated for each child of $\mathcal{E}_f$ and $\mathcal{E}_g$ (lines 7 and 8). We will describe this for $\mathcal{E}_f$ as input to algorithm 2: in the case that the order of $\mathcal{E}_f$ is greater or equal to the order of $\mathcal{E}_g$ (line 3 of algorithm 2), the input weight of $\mathcal{E}_f$ is pushed down along the edges, i.e. for each edge (χ, w) an EVMDD with $\kappa + w$ as input weight and χ as root node is returned. If the order of $\mathcal{E}_f$ is lower (line 4 of algorithm 2), $\mathcal{E}_f$ is just copied $k+1$ times, where $k + 1$ denotes the number of children of $\mathcal{E}_g$. This case can be seen as some sort of alignment, i.e. we push down the weights along the edges of the higher ordered EVMDD only, until both EVMDDs are aligned on the same level.

Back in the main algorithm, line 10 recursively calls $apply$ for each pair of the returned EVMDDs. Finally, in line 12 the final EVMDD $\mathcal{E}_h$ is constructed. The input weight val of this EVMDD is the minimum input weight of all EVMDDs returned by the recursive call. The root node $\mathbf{h}$ of $\mathcal{E}_h$ is associated with

Algorithm 1: APPLY algorithm for two EVMDDs.

Function APPLY$(\circ, \mathcal{E}_f = \langle \kappa_f, \mathbf{f} \rangle, \mathcal{E}_g = \langle \kappa_g, \mathbf{g} \rangle)$
  if terminal_case$(\mathcal{E}_f, \mathcal{E}_g, \circ)$ **then**
    return $\mathcal{E}_h = (\kappa_f \circ \kappa_g, \mathbf{0})$
  /* process $\mathcal{E}_f$ and $\mathcal{E}_g$ according to their ordering */
  $max\_level = \max(level(\mathrm{var}(\mathbf{f})), level(\mathrm{var}(\mathbf{g})))$
  $v = \mathrm{var}(max\_level)$
  $k = |\mathcal{D}_v| - 1$
  $\mathcal{E}_f^0, \ldots, \mathcal{E}_f^k =$ SUB_EVMDDS$(\mathcal{E}_f, max_level, k)$
  $\mathcal{E}_g^0, \ldots, \mathcal{E}_g^k =$ SUB_EVMDDS$(\mathcal{E}_g, max_level, k)$
  /* recursive call for each child */
  foreach $i \in \{0, \ldots, k\}$ **do**
    $\langle \kappa_h^i, \mathbf{h}^i \rangle =$APPLY$(\circ, \mathcal{E}_f^i, \mathcal{E}_g^i)$
  /* normalize resulting EVMDD $\mathcal{E}_h$ */
  $val = \min(\kappa_h^1, \ldots, \kappa_h^k)$
  $\mathcal{E}_h = \langle val, (v, \mathbf{h}^0, \ldots, \mathbf{h}^k, \kappa_h^0 - val, \ldots, \kappa_h^k - val) \rangle$
  /* reduce EVMDD $\mathcal{E}_h$ */
  $\mathcal{E}_h = \mathrm{reduce}(\mathcal{E}_h)$
  return $\mathcal{E}_h$

Algorithm 2: Algorithm to generate sub-EVMDDs.

Function
SUB_EVMDDS$(\mathcal{E} = \langle \kappa, \mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k) \rangle, max_level, k)$
  /* $\mathcal{E}$ is ordered on the same level or before the other EVMDD */
  if $level(v) = max\_level$ **then**
    return $\langle \kappa + w_0, \chi_0 \rangle, \ldots, \langle \kappa + w_k, \chi_k \rangle$
  /* $\mathcal{E}$ is ordered after the other EVMDD */
  return $\underbrace{\mathcal{E}, \ldots, \mathcal{E}}_{k+1 \text{ times}}$

the root node variable of the higher ordered EVMDD, the children of $\mathbf{h}$ are the EVMDDs returned by the recursive call, and the weights are the input weights of these EVMDDs minus the minimum value. As a result, $\mathcal{E}_h$ will be *normalized* in the sense that the minimum weight of all children is 0. Before $\mathcal{E}_h$ is returned it gets reduced (line 13). If we only require quasi-reducedness, we could change the reduction algorithm, however, quasi-reduced EVMDDs result in more nodes and therefore it might be more efficient to only quasi-reduce the final resulting EVMDD.

One important note we have to make is that usually a cache is used to store $apply$ results for functions f, g with operator $\circ$. This cache is checked after line 3, to immediately return the resulting EVMDD if it was previously computed. Without this cache, $apply$ might have an exponential number of recursive calls, even if both input EVMDDs and the resulting EVMDD are only polynomial in size.[3] If a cache is used, we can further modify the algorithm (Lai et al. 1996) and check after the recursive call (line 10) if all EVMDDs $\langle \kappa_h^i, \mathbf{h^i} \rangle$ have the same input weight and the same root node. This would result in a node where all edges have weight 0 and lead to the same successor node, and such a node would get reduced later on. Instead of generating this node, we can immediately return an EVMDD where the root node is $\mathbf{h}^0$ and the input weight is κ_h^0. With this modification, and if a cache is used, an EVMDD is already reduced before line 13 and we can omit the reduction call. Furthermore, similarly to the EVBDD case, the code has to be slightly adapted if one wants to exploit operators $\circ$ which satisfy the additive property $(\kappa_f + f) \circ (\kappa_g + g) = (\kappa_f \circ \kappa_g) + (f \circ g)$. In this case, the complexity of the $apply$ operation is polynomial in the size of $\mathcal{E}_f$ and $\mathcal{E}_g$ (Lai et al. 1996).

The $apply$ algorithm allows us to generate EVMDDs for functions that are represented in a syntactic form, e.g. as terms of multivariate polynomials. Such terms can be represented as an abstract syntax tree (AST). Before we give an example of this, we want to mention that we can also construct EVMDDs by applying Boole's expansion theorem in a top-down fashion (Mattmüller et al. 2017). Additionally, if the function is given in tabular form, i.e. each row in the table represents a state and its cost, we can construct the path corresponding to each state, assign the cost to the edge connected to the terminal node, and reduce the diagram after all state paths have been constructed (Hooker 2013).

Example 4. Consider the cost function for a drive action from the logistics task in Example 1. For this example, we will use the formal domain definition with $\mathcal{D}_{\text{p}_1\text{-at}} = \mathcal{D}_{\text{p}_2\text{-at}} = \{0, 1, 2, 3\}$, where the numbers correspond to the previous domain values. Then we have $c(s) = [\text{p}_1\text{-at} = 3] + [\text{p}_2\text{-at} = 3] + 1$. Note that we treat $=$ as a binary operator with $(f = g)(s) = 1$ if $f(s) = g(s)$, and $(f = g)(s) = 0$, otherwise. Figure 2.8 illustrates the construction process.

[3] In the literature, this cache is often introduced "for efficiency reasons". While this is certainly true, it may not completely reflect the importance of this cache.

The AST of c is illustrated in grey. EVMDDs generated during the construction process are illustrated in red (to reduce clutter terminal nodes are depicted as circles). For example, the term $\mathrm{p}_2\text{-at} = 3$ generates an EVMDD for $\mathrm{p}_2\text{-at}$, where each edge weight is set to the domain value of the corresponding edge, and an EVMDD for 3, which consists of a single node with input weight 3. Then, the apply algorithm is called for operator $=$.

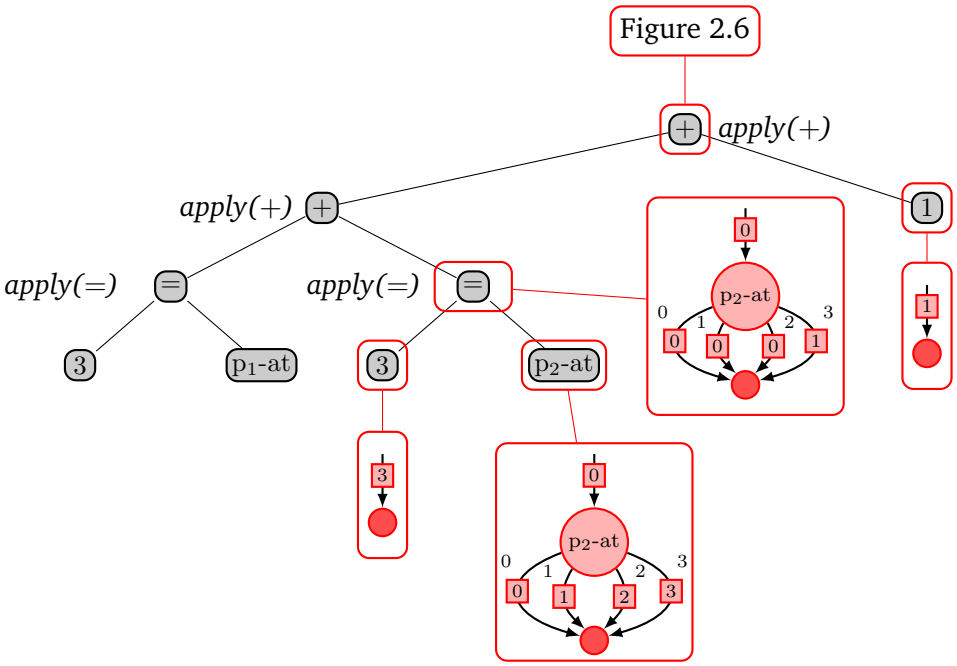


Figure 2.8: AST of cost function $[\mathrm{p}_1\text{-at} = 3] + [\mathrm{p}_2\text{-at} = 3] + 1$ is illustrated in grey. EVMDDs generated during the construction process are illustrated in red. Figure 2.6 is the previously shown final EVMDD with named domain values.

2.3.6 EVMDDs and Cartesian sets

We will frequently discuss planning techniques which are based on Cartesian sets of states. Most of these techniques are used to compute admissible heuristic estimates and therefore we are often required to compute the minimum cost of an action a applicable in a Cartesian set of states s_C.

Definition 18 (Cartesian action cost). Let s_C be a Cartesian set of states and a an action. Then, $cost_a(s_C) = \min_{s \in s_C} c_a(s)$.

Interestingly, an action a applicable in s_C is always applicable in a state which minimizes the cost function.

Lemma 3. *Let s_C be a Cartesian set of states and a an action such that a is applicable in s_C. Let $s_{\min}$ be the set of states which minimizes the cost in s_C, i.e. $s_{\min} = \{s | s = \operatorname{argmin}_{s' \in s_C} c_a(s)\}$. Then, a is applicable in some state in $s_{\min}$.*

Proof. Follows from the definition of Cartesian sets and Assumption 3. Since s_C is Cartesian it is of the form $D_1 \times \ldots \times D_n$. We can construct the subset of these states $s_C|\mathrm{pre}$ such that a is always applicable in each $s \in s_C|\mathrm{pre}$ by restricting D_i to $\mathrm{pre}(a)$ where $\mathrm{pre}(a)$ is defined. Now, from Assumption 3 follows that $\mathrm{vars}(c_a) \cap \mathrm{vars}(\mathrm{pre}(a)) = \emptyset$, which means that s_C agrees with $s_C|\mathrm{pre}$ on the domain values defined over $\mathrm{vars}(c_a)$. In particular, the valuation which minimizes the cost also appears in $s_C|\mathrm{pre}$. Thus, $s_{\min} \cap s_C|\mathrm{pre} \neq \emptyset$, concluding the proof. □

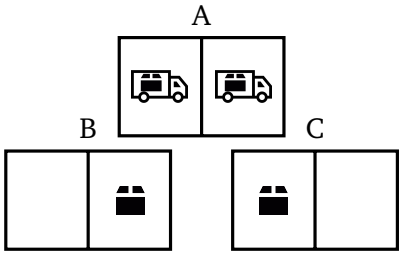
In principle, we could compute the minimum cost by computing the cost of a in each state contained in s_C. However, for Cartesian sets which subsume exponentially many concrete states this is infeasible. It turns out that we can make use of the corresponding cost function EVMDD to efficiently compute the minimum cost given a Cartesian set of states, by locally minimizing the paths in the EVMDD. Figure 2.9 depicts the underlying intuition. On the left we have the Cartesian set of states $s_C = \{A\} \times \{C, t\} \times \{B, t\}$, i.e. the truck is in all states at position A, but package 1 is either at location C or in the truck, and package 2 is either at location B or in the truck. On the right we have the EVMDD of the cost function of a drive action. Paths which are consistent with a state in s_C are marked red. Obviously, evaluating each *path* separately corresponds to evaluating each state in s_C. But observe that instead of evaluating the EVMDD *globally* we can also evaluate the EVMDD *locally*, i.e. we first evaluate edges of the node corresponding to $\mathrm{p_1\text{-}at}$, choose the edge with minimum weight, and then choose the cheapest outgoing edge of the node corresponding to $\mathrm{p_2\text{-}at}$.

Definition 19 (Local EVMDD minimization). Let s_C be a Cartesian set of states, a an action and $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ the EVMDD of c_a. The arithmetic function of $\mathcal{E}$ in terms of *local minimization* over s_C is given by $\kappa + f^L$, where f^L is the function denoted by $\mathbf{f}$. The terminal node $\mathbf{0}$ denotes the constant function 0, and $(v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ denotes the arithmetic function given by

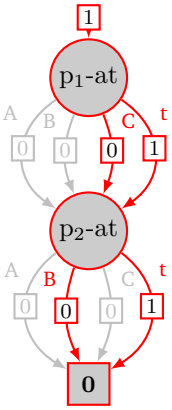
$$f^L(s_C) = \min_{d \in s_C(v)} (f^L_d(s_C) + w_d),$$

where $f^L_d(s_C)$ is the arithmetic function denoted by child χ_d. We write $\mathcal{E}^L(s_C)$ for $\kappa + f^L(s_C)$.

Theorem 3. *Let s_C be a Cartesian set of states, a an action and $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ the EVMDD of c_a. Then $c_a(s_C) = \mathcal{E}^L(s_C)$.*



(a) Cartesian set of states s_C.



(b) EVMDD of the drive cost function, paths consistent with s_C are marked red.

Figure 2.9

Proof. By definition, $c_a(s_C) = \min_{s \in s_C} c_a(s) = \min_{s \in s_C} \mathcal{E}(s)$. We now show by induction over the depth of $\mathcal{E}$ that $\min_{s \in s_C} \mathcal{E}(s) = \mathcal{E}^L(s_C)$. For the base case, $\mathbf{f} = \mathbf{0}$, we have $\min_{s \in s_C} \mathcal{E}(s) = \kappa + 0 = \mathcal{E}^L(s_C)$. Let $v = \mathrm{var}(\mathbf{f})$ and assume the assumption holds for nodes below $\mathbf{f}$, i.e.

$$\min_{s \in s_C} \left(\mathcal{E}_{s(v')}(s) + w_{s(v')} \right) = \min_{d \in s_C(v')} (f_d^L(s_C) + w_d),$$

for $v' \neq v$. By definition of $\mathcal{E}(s)$ we have

$$\begin{aligned} \min_{s \in s_C} \mathcal{E}(s) &= \min_{s \in s_C} (\kappa + f(s)) \\ &= \kappa + \min_{s \in s_C} f(s) \\ &= \kappa + \min_{s \in s_C} f_{s(v)}(s) + w_{s(v)}. \end{aligned}$$

Since s_C is Cartesian it is of the form $D_1 \times \ldots \times D_n$. Let i be the index of the corresponding domain of v, i.e. $D_i \subseteq \mathcal{D}_v$. Then, we get

$$
\begin{aligned}
&\min_{s \in s_C} f_{s(v)}(s) + w_{s(v)} \\
&= \min_{d_1,\ldots,d_n \in \prod_{j\in\{1,\ldots,n\}} D_j} \left(f_{d_i}((v_1 \doteq d_1) \wedge \ldots \wedge (v_n \doteq d_n)) + w_{d_i} \right) \\
&= \min_{d_i \in D_i} \min_{d_1,\ldots,d_n \in \prod_{j\in\{1,\ldots,n\}\setminus i} D_j} \left(f_{d_i}((v_1 \doteq d_1) \wedge \ldots \wedge (v_n \doteq d_n)) + w_{d_i} \right) \\
&= \min_{d \in s_C(v)} \left(f_{d_i}(s_C) + w_{d_i} \right),
\end{aligned}
$$

where the last equation holds because the value of d_i does not matter for evaluation of $f_{d_i}(s_C)$. From the induction assumption we then get

$$
\min_{d \in s_C(v)} \left(f_{d_i}(s_C) + w_{d_i} \right) = \min_{d \in s_C(v)} \left(f^L_{d_i}(s_C) + w_{d_i} \right),
$$

concluding the proof. □

Thus, when we are required to compute the cost of an action for a Cartesian set of states we can locally minimize the corresponding EVMDD $\mathcal{E}$. This minimization is linear in the size of $\mathcal{E}$. Therefore, if $\mathcal{E}$ is small computing the cost for a Cartesian set is efficient. We will later see examples where this matters. Also note that the local minimization corresponds to computing a cheapest path from the root node to the terminal node, by use of Dijkstra's algorithm (Dijkstra 1959).

In the following, we give an example for a non-Cartesian set of states and why local minimization is not precise in this case.

Example 5. Let us again assume that the logistics task only contains variables $\mathrm{p_1\text{-}at}$ and $\mathrm{p_2\text{-}at}$ and let $S = \{(\mathrm{p_1\text{-}at} \doteq A) \wedge (\mathrm{p_2\text{-}at} \doteq t), (\mathrm{p_1\text{-}at} \doteq t) \wedge (\mathrm{p_2\text{-}at} \doteq A)\}$. Clearly, S is not Cartesian. Figure 2.10 depicts the EVMDD of the drive action and the corresponding paths in the EVMDD. Local minimization would yield a cost of 1, but applying the cost function in any state of S induces a cost of 2.

We finish this section about decision diagrams by introducing a second form of EVMDDs, which we have already seen before under another name.

2.3.7 Flattened EVMDDs

To analyze the time complexity of operations based on EVBDDs, Lai (1993) introduces (reduced ordered) *flattened EVBDDs*. The idea behind flattened EVBDDs is that the edge weights are pushed all the way down to the terminal node. As several paths may have different sums of edge weights, flattened EVBDDs consist of multiple terminal nodes, each terminal node is associated with an integer, corresponding to the cost of a (or multiple) path(s) in the

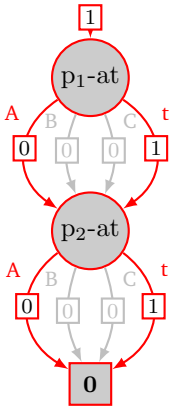


Figure 2.10: EVMDD of the drive cost function, paths consistent with S are marked red.

EVBDD. As a consequence, flattened EVBDDs are analogous to ADDs. We now introduce *flattened EVMDDs*, which are the natural generalization of flattened EVBDDs to multi-valued variables, and are thus analogous to MTMDDs. We use the term flattened EVMDD since it often allows a simpler view on the problem, if the EVMDD representation is already given.

Definition 20 (Flattened EVMDD). A *flattened EVMDD* $\mathcal{E}_f^{\mathcal{F}} = \mathbf{f}$ over a finite set of variables $\mathcal{V}$ is a directed acyclic graph consisting of two types of nodes: (1.) A *nonterminal node* $\mathbf{v}$ is a tuple $(v, \chi_0, \ldots, \chi_k)$ where $v \in \mathcal{V}$ is a variable, $k = |\mathcal{D}_v| - 1$, and the children $\chi_0, \ldots, \chi_k$ are terminal or nonterminal nodes of $\mathcal{E}_f^{\mathcal{F}}$. (2.) A non-terminal node $\mathbf{w} = w$ is associated with an integer $w \in \mathbb{Q}^+$.

Again, $\mathbf{f}$ is the root node of $\mathcal{E}_f^{\mathcal{F}}$. Reduced, ordered, flattened EVMDDs are defined in the same way as reduced ordered EVMDDs.

Definition 21 (Arithmetic function denoted by the flattened EVMDD). A flattened EVMDD $\mathcal{E}_f^{\mathcal{F}} = \mathbf{f}$, denotes the arithmetic function f. A terminal node $\mathbf{w} = w$ denotes the constant function w. A non-terminal node $(v, \chi_0, \ldots, \chi_k)$ denotes the arithmetic function over $\mathcal{S}$ given by $f(s) = f_{s(v)}(s)$, where $f_{s(v)}$ is the arithmetic function denoted by child $\chi_{s(v)}$.

Example 6. Consider again the cost function of the drive actions of the logistics task presented in Example 1: $c_{\text{drive-AB}} = [\text{p}_1\text{-at} = t] + [\text{p}_2\text{-at} = t] + 1$. Figure 2.11 depicts the flattened EVMDD representation of $c_{\text{drive-AB}}$.

A flattened EVMDD representing function f has as many terminal nodes as values in the domain of f. While flattened EVMDDs can have exponentially more nodes than the corresponding EVMDD, we will see that it can sometimes

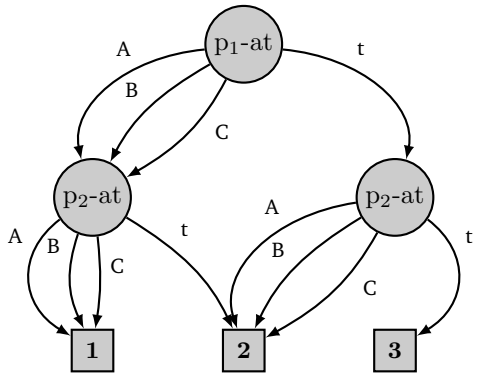


Figure 2.11: Flattened EVMDD of the drive cost function $[\mathrm{p}_1\text{-at} = t] + [\mathrm{p}_2\text{-at} = t] + 1$.

be beneficial to represent cost functions with flattened EVMDDs, with one caveat: the efficient local minimization we gain from EVMDDs is not possible with flattened EVMDDs, since all edge weights are 0. Thus, if the minimization plays a critical role, flattened EVMDDs are inferior to plain EVMDDs, even if they would have the same size.

2.3.8 Other types of decision diagrams

There also exist a few other types of decision diagrams, which are now briefly mentioned here. *Affine Algebraic Decision Diagrams* (AADDs), introduced by Sanner and McAllester (2005) for probabilistic reasoning, can be seen as a generalization of EVMDDs. They allow to compactly represent functions which have additive *and* multiplicative structure, but their evaluation method is more involved than that of EVMDDs, which makes them less straightforward to apply to our setting.

All of the above decision diagrams relied on Boole's expansion theorem but there also exist decision diagrams which rely on other types of expansion. *Functional Decision Diagrams* (FDDs) (Kebschull et al. 1992) rely on the (positive and negative) Davio expansion (also called the Reed-Muller expansion (Muller 1954; Reed 1954)) and are more suitable to represent XOR-intensive functions. Becker et al. (1995) show that there are classes of functions for which BDDs are exponentially more compact than FDDs, but also the other way around. Furthermore, Becker and Drechsler (1995) show that these expansion types are the only types that help to reduce the size of decision diagrams (if another technique, *Complemented Edges* (Brace et al. 1990) is additionally applied). Due to this, Drechsler and Becker (1998) propose Kronecker Functional Decision Diagrams (KFDDs), which are based on both Shannon and Davio expansion, to combine the advantages of BDDs and FDDs. While it would be interesting to see how these types of decision diagrams can

be applied to our setting, we settle for now on representing cost functions with EVMDDs.

2.4 Planning Task Compilations

Now that we have a representation of state-dependent action costs which exhibits the structure underlying the cost function, we will look at different ways to make use of this representation during the planning process. Much of the research in AI planning is concerned with classical planning and heuristic search, and we will see that the generalization of many heuristics is not straightforward applied to tasks with state-dependent action costs. One way to circumvent the definition and analysis of the generalized version of such heuristics is to compile state-dependent action costs away, i.e. transform the task into an equivalent problem where the costs of actions are constant, and apply well known classical planning heuristics to the compiled problem. We first formally define the compilation of a planning task.

Definition 22 (Compilation). Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_I, s_\star, c \rangle$ and $\Pi' = \langle \mathcal{V}', \mathcal{A}', s'_I, s'_\star, c' \rangle$ be two planning tasks. A compilation is a tuple of functions $\mathcal{C} = \langle \mathcal{C}_\mathcal{V}, \mathcal{C}_\mathcal{A}, \mathcal{C}_\mathcal{S}, \mathcal{C}_\star, \mathcal{C}_c \rangle$ that induces a mapping from Π to Π', such that $\mathcal{V}' = \mathcal{C}_\mathcal{V}(\Pi)$, $\mathcal{A}' = \mathcal{C}_\mathcal{A}(\Pi)$, $s'_I = \mathcal{C}_\mathcal{S}(\Pi, s_I)$, $s'_\star = \mathcal{C}_\star(\Pi)$, and $c' = \mathcal{C}_c(\Pi)$.

Given a compilation $\mathcal{C}$ we denote Π' with $\Pi^\mathcal{C}$ and say $\Pi^\mathcal{C}$ is the *compilation of* Π *under* $\mathcal{C}$, or simply compilation of Π. If it is clear from the context, we will also simply write $\mathcal{C}(s)$ instead of $\mathcal{C}_\mathcal{S}(\Pi, s)$. We will often examine how different heuristics behave under compilation. For this, the notion of heuristic invariance under compilation will be important. Informally, a heuristic is invariant under compilation if it produces the same estimates in the original and the compiled task.

Definition 23 (Heuristic invariance under compilation). Let Π be a planning task and $\mathcal{C}$ a compilation. A heuristic h is *invariant* under $\mathcal{C}$ if $h(s) = h(\mathcal{C}_\mathcal{S}(s))$ for all $s \in \mathcal{S}$.

In the following, we will define a compilation from a task with state-dependent action costs to a classical planning task which is based on a commonly known compilation of conditional effects.

2.4.1 Exponential Compilation

The first type of compilation, proposed by Ivankovic et al. (2014), is based on a compilation of a planning task with conditional effects into a classical planning task. A planning task with conditional effects may have actions where some effects of an action only occur in certain states (specified by the effect condition). Conditional effects can be compiled away with exponential

overhead (Gazen and Knoblock 1997) by introducing new actions for each possible subset of conditional effects, and appending the effect conditions to the action precondition. For planning tasks with state-dependent action costs the corresponding compilation thus generates new actions for each possible assignment of variables in the support of the cost function, and the cost of an action is the function evaluation under this assignment.

Definition 24 (Exponential compilation). Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_I, s_\star, c \rangle$ be a planning task. Let $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ be an action and p a partial variable assignment. Then $a^p = \langle \mathrm{pre} \cup p, \mathrm{eff} \rangle$. Now, let $P(c_a)$ be a set of partial variable assignments such that $P(c_a) = \{p | \mathrm{vars}(p) = \mathrm{vars}(c_a), \text{and} \, \forall v \in \mathrm{vars}(c_a) : p(v) \in \mathcal{D}_v\}$, i.e. the set of consistent variable assignments over the support of c_a. Then $\mathrm{EXP}(a) = \{a^p | p \in P(c_a)\}$ and we define $\mathrm{EXP}(c)$ such that $\mathrm{EXP}(c)(a^p, s) = c_a(p)$ for all $s \in \mathcal{S}$.

The *exponential compilation* of Π, denoted as Π^{EXP}, is the compilation with $\mathrm{EXP}_{\mathcal{V}}(\Pi) = \mathcal{V}$, $\mathrm{EXP}_{\mathcal{S}}(s) = s$, $\mathrm{EXP}_{s_\star}(\Pi) = s_\star$, $\mathrm{EXP}_{\mathcal{A}}(\Pi) = \{a' | a' \in \mathrm{EXP}(a) \text{ for all } a \in \mathcal{A}\}$, and $\mathrm{EXP}_c(\Pi)$ as above.

Example 7. The key idea of the exponential compilation is to have a constant cost action for each possible assignment of the action cost function and add this assignment to the precondition of the new action. Let us consider the drive actions of our logistics planning task. We have an action $\mathrm{drive\text{-}AB} = \langle (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle$ with cost function $[\mathrm{p}_1\text{-at} = t] + [\mathrm{p}_2\text{-at} = t] + 1$. Since the support consists of two variables with four domain values each, we get 16 possible partial variable assignments. To ease illustration, we write a instead of $\mathrm{drive\text{-}AB}$. Exponential compilation then results in the following actions:

$$
\begin{aligned}
a^{(\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq A)} &= \langle (\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq A) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle \\
a^{(\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq B)} &= \langle (\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq B) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle \\
a^{(\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq C)} &= \langle (\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq C) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle \\
a^{(\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq t)} &= \langle (\mathrm{p}_1\text{-at} \doteq A) \wedge (\mathrm{p}_2\text{-at} \doteq t) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle \\
a^{(\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq A)} &= \langle (\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq A) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle \\
a^{(\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq B)} &= \langle (\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq B) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle \\
a^{(\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq C)} &= \langle (\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq C) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle \\
a^{(\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq t)} &= \langle (\mathrm{p}_1\text{-at} \doteq B) \wedge (\mathrm{p}_2\text{-at} \doteq t) \wedge (\text{t-at} \doteq A), (\text{t-at} \doteq B) \rangle
\end{aligned}
$$

$$
\begin{aligned}
a^{(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq A)} &= \langle(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq A)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle\\
a^{(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq B)} &= \langle(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq B)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle\\
a^{(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq C)} &= \langle(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq C)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle\\
a^{(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq t)} &= \langle(\text{p}_1\text{-at}\doteq C)\wedge(\text{p}_2\text{-at}\doteq t)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle\\
a^{(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq A)} &= \langle(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq A)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle\\
a^{(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq B)} &= \langle(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq B)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle\\
a^{(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq C)} &= \langle(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq C)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle\\
a^{(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq t)} &= \langle(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq t)\wedge(\text{t-at}\doteq A),(\text{t-at}\doteq B)\rangle,
\end{aligned}
$$

and the action cost is either 1 for actions based on partial assignments where no package is in the truck, 2 for actions based on assignments where a single package is in the truck, and 3 for action $\text{drive-AB}^{(\text{p}_1\text{-at}\doteq t)\wedge(\text{p}_2\text{-at}\doteq t)}$. Similarly, we get 16 actions for each of the other drive actions, resulting in a total number of $4 + 6 \cdot 16 = 100$ actions in the compiled task.

Note that the exponential compilation of Π shares the same state space as Π. Additionally, it preserves the transition system induced by Π.

Lemma 4 (Exponential compilation preserves transitions). *Let Π be a planning task. There is a transition $s \xrightarrow{a,c_a(s)} s[a]$ in $\mathcal{T}(\Pi)$ with label a and weight $c_a(s)$, if and only if there is a transition $s \xrightarrow{a',c_{a'}(s)} s[a']$ in $\mathcal{T}(\Pi^{\text{EXP}})$ with label a' and weight $c_{a'}(s)$, and $c_{a'}(s) = c_a(s)$.*

Proof. Follows from the definition of Π^{EXP}: set $a' = a^p$, such that $c_a(p) = c_a(s)$. □

Corollary 1. *Let Π be a planning task, then:*

1. *Π^{EXP} is a classical planning task,*
2. *$|\mathcal{A}^{\text{EXP}}| = \sum_{a\in\mathcal{A}} \prod_{v\in\text{vars}(c_a)} |\mathcal{D}_v|$,*
3. *There exists a plan $\pi = \langle a_0, \ldots, a_{n-1}\rangle$ for Π, if and only if there exists a plan $\pi' = \langle a'_0, \ldots, a'_{n-1}\rangle$ for Π^{EXP}, with $a'_i \in \text{EXP}(a_i)$ for $i \in \{0, \ldots, n-1\}$, and $c_{a_i}(s_I[a_0]\ldots[a_{i-1}]) = c_{a'_i}(s_I[a'_0]\ldots[a'_{i-1}])$. Therefore $cost(\pi) = cost(\pi')$.*

Proof. The first statement follows from the construction of Π^{EXP}. For the second, there are $|P(c_a)|$ actions for each $a \in \mathcal{A}$ and $|P(c_a)| = \prod_{v\in\text{vars}(c_a)} |\mathcal{D}_v|$, as we enumerate all possible valuations of variables in the support. For the last statement, consider that the transition systems of both tasks share the same structure. □

While the exponential compilation is not a practical approach for dealing with state-dependent action costs, it serves as a useful tool to theoretically evaluate other types of compilation, or to evaluate the generalization of classical planning approaches, as an (optimal) plan for the exponential compilation can be transformed to an (optimal) plan for the original task.

2.4.2 EVMDD-based Compilation

The next type of compilation is based on EVMDDs and is polynomial in the size of the largest action cost EVMDD. Therefore, if our cost functions have a compact representation, then this compilation will be compact as well. The crucial insight is that edges in the EVMDD can be thought of as *auxiliary actions* with *state-independent* constant costs that are only allowed to be applied if the tested state variable matches the value corresponding to the edge. To that end, we introduce an additional auxiliary variable aux^a for each original action a that keeps track of where we currently stand in the evaluation of the corresponding EVMDD.

Definition 25 (EVMDD-based action compilation). Let $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ be an action with cost function c_a and let $\mathcal{E}_{c_a} = \langle \kappa, \mathbf{f} \rangle$ be a corresponding EVMDD representation. Let $idx : \mathbf{v} \to \{1, \ldots, |\mathcal{E}_{c_a}|\}$ be a topological ordering of $\mathcal{E}_{c_a}$, i.e. a bijective function numbering the nodes of $\mathcal{E}_{c_a}$ such that for every non-terminal node $\mathbf{v}$ and each child $\chi_i(\mathbf{v}) : idx(\mathbf{v}) < idx(\chi_i(\mathbf{v}))$, and $idx(\mathbf{0}) = |\mathbf{f}|$.

Let aux^a be a variable with $\mathcal{D}_{\mathrm{aux}^a} = \{0, \ldots, |\mathbf{f}|\}$. For each $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ and each $i = 0, \ldots, k$ we get an action

$$a^{(v \doteq i), idx(\mathbf{v})} = \langle (\mathrm{aux}^a \doteq idx(\mathbf{v})) \wedge (v \doteq i), (\mathrm{aux}^a \doteq idx(\chi_i(\mathbf{v}))) \rangle$$

with cost $c(a^{(v \doteq i), idx(\mathbf{v})}, s) = w_i$ for all $s \in \mathcal{S}$. We denote the set of actions induced by $\mathbf{v}$ as $a^{\mathbf{v}} = \{a^{(v \doteq i), idx(\mathbf{v})} | i = 0, \ldots, k\}$ and the set of actions induced by $\mathcal{E}_{c_a}$ as $a^{\mathcal{E}} = \bigcup_{\mathbf{v} \in \mathcal{I}} a^{\mathbf{v}}$, where $\mathcal{I}$ denotes the set of non-terminal nodes in $\mathcal{E}_{c_a}$.

Additionally, we require an initial action which corresponds to the precondition of the original action and initiates the chain of EVMDD actions, and a concluding action, which applies the effects of the action. For this, we introduce a global semaphore variable lock, to ensure that we don't interweave two EVMDD evaluations (of two different actions) at once and instead process EVMDD paths sequentially.

Definition 26 (Initial and concluding action). Let $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ be an action and lock a binary variable. Then we have an *initial action* a^{pre} and a *concluding action* a^{eff} with

$$a^{\mathrm{pre}} = \langle \mathrm{pre} \wedge (\mathrm{lock} \doteq 0) \wedge (\mathrm{aux}^a \doteq 0), (\mathrm{lock} \doteq 1) \wedge (\mathrm{aux}^a \doteq 1) \rangle$$
$$a^{\mathrm{eff}} = \langle (\mathrm{aux}^a \doteq |\mathbf{f}|), \mathrm{eff} \wedge (\mathrm{aux}^a \doteq 0) \wedge (\mathrm{lock} \doteq 0) \rangle$$

with cost $c(a^{\text{pre}}, s) = \kappa$, $c(a^{\text{eff}}, s) = 0$, for all states $s \in \mathcal{S}$.

Example 8. Let us apply the EVMDD compilation to the action $\text{drive-AB} = \langle(\text{t-at} \doteq A), (\text{t-at} \doteq B)\rangle$ from our logistics task. The EVMDD of $c_{\text{drive-AB}}$ is depicted in Figure 2.12 and is the same as in Figure 2.6. The domain of the auxiliary variable $\text{aux}^{\text{drive-AB}}$ is $\{0, 1, 2, 3\}$ since the EVMDD consists of 3 nodes. In the following, we write a for drive-AB and aux for $\text{aux}^{\text{drive-AB}}$, and we get the following actions:

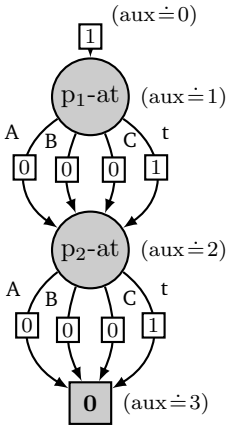


Figure 2.12: EVMDD of the drive-AB cost function annotated with corresponding auxiliary variable values in the EVMDD compilation.

$$
\begin{aligned}
a^{\text{pre}} &= \langle(\text{t-at} \doteq A) \wedge (\text{lock} \doteq 0) \wedge (\text{aux} \doteq 0), (\text{lock} \doteq 1) \wedge (\text{aux} \doteq 1)\rangle \\
a^{(\text{p}_1\text{-at} \doteq A),1} &= \langle(\text{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq A), (\text{aux} \doteq 2)\rangle \\
a^{(\text{p}_1\text{-at} \doteq B),1} &= \langle(\text{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq B), (\text{aux} \doteq 2)\rangle \\
a^{(\text{p}_1\text{-at} \doteq C),1} &= \langle(\text{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq C), (\text{aux} \doteq 2)\rangle \\
a^{(\text{p}_1\text{-at} \doteq t),1} &= \langle(\text{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq t), (\text{aux} \doteq 2)\rangle \\
a^{(\text{p}_2\text{-at} \doteq A),2} &= \langle(\text{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq A), (\text{aux} \doteq 3)\rangle \\
a^{(\text{p}_2\text{-at} \doteq B),2} &= \langle(\text{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq B), (\text{aux} \doteq 3)\rangle \\
a^{(\text{p}_2\text{-at} \doteq C),2} &= \langle(\text{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq C), (\text{aux} \doteq 3)\rangle \\
a^{(\text{p}_2\text{-at} \doteq t),2} &= \langle(\text{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq t), (\text{aux} \doteq 3)\rangle \\
a^{\text{eff}} &= \langle(\text{aux} \doteq 3), (\text{t-at} \doteq B) \wedge (\text{lock} \doteq 0) \wedge (\text{aux} \doteq 0)\rangle
\end{aligned}
$$

We have an action cost of 1 for $a^{(\mathrm{p}_1\text{-at}\doteq t),1}$, $a^{(\mathrm{p}_2\text{-at}\doteq t),2}$ and a^{pre}. These correspond to edges in the EVMDD with a weight of 1. All other actions have a cost of 0.

Definition 27 (EVMDD compilation). Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_I, s_\star, c\rangle$ be a planning task. The *EVMDD compilation* of Π, denoted as Π^{DD}, is the compilation where

- $\mathrm{DD}_{\mathcal{V}}(\Pi) = \mathcal{V} \cup \{\mathrm{lock}\} \cup \{\mathrm{aux}^a | a \in \mathcal{A}\}$
- $\mathrm{DD}_{\mathcal{A}}(\Pi) = \bigcup_{a\in\mathcal{A}} \{a^{\mathrm{pre}}\} \cup a^{\mathcal{E}} \cup \{a^{\mathrm{eff}}\}$,
- $\mathrm{DD}_{\mathcal{S}}(s) = s \wedge (\mathrm{lock} \doteq 0) \bigwedge_{a\in\mathcal{A}} (\mathrm{aux}^a \doteq 0)$,
- $\mathrm{DD}_{s_\star}(\Pi) = s_\star \wedge (\mathrm{lock} \doteq 0) \bigwedge_{a\in\mathcal{A}} (\mathrm{aux}^a \doteq 0)$,
- and costs as above.

We will often write s^{DD} for $\mathrm{DD}_{\mathcal{S}}(s)$. If $\mathcal{A}$ consists of state-dependent and constant cost actions, then we can also only compile actions with state-dependent cost, and preserve constant cost actions.

The semaphore variable prevents evaluating multiple EVMDDs (i.e. costs) concurrently. Without a semaphore variable, one could concurrently activate two action sequences, where the concluding effect of the first sequence increases the cost of the second sequence. Before this concluding effect is enabled, the second action sequence can be triggered with cheaper costs than possible. Consider the following example:

Example 9. Consider a planning task with binary variables x and y and two actions $a_x = \langle \top, (x \doteq 1)\rangle$ with $c_{a_x} = y$ and $a_y = \langle \top, (y \doteq 1)\rangle$ with $c_{a_y} = x$. Consider state $s = (x \doteq 0) \wedge (y \doteq 0)$. After application of one action, the other (distinct) action will have a cost of 1. Now consider the EVMDD compilation, but without a semaphore variable. Then, beginning from s, the sequence of actions $\langle a_x^{\mathrm{pre}}, a_x^{(y\doteq 0),1}, a_y^{\mathrm{pre}}, a_y^{(x\doteq 0),1}, a_x^{\mathrm{eff}}, a_y^{\mathrm{eff}}\rangle$ is possible, with a total cost of 0. With a semaphore variable, it is not possible to apply $a_y^{(x\doteq 0),1}$ between a_x^{pre} and a_x^{eff}, and every sequence of (distinct) actions induces at least a cost of 1.

Note that we could also use the input weight as the cost of the concluding action, instead of the cost of the initial action. However, putting the unconditional cost in front of the action sequence can be helpful for search, where the cost may already disqualify the initial action from being considered. However, as we will mostly use compilation as a means to produce heuristic values, this will not be of importance to us.

While EVMDD-based compilation does not preserve plan length, it preserves plan costs.

Lemma 5 (EVMDD compilation preserves costs). *Let Π be a planning task. There is a transition $s \xrightarrow{a,c_a(s)} s[a]$ in $\mathcal{T}(\Pi)$ with label a and weight $c_a(s)$, if and only if there is a sequence of transitions in $\mathcal{T}(\Pi^{\mathrm{DD}})$ starting in $\mathrm{DD}_{\mathcal{S}}(s)$ and ending in $\mathrm{DD}_{\mathcal{S}}(s')$, and the sum of the weights is exactly $c_a(s)$.*

Proof. We first show sufficiency. Let there be a transition $s \xrightarrow{a,c_a(s)} s[a]$ with label a and weight $c_a(s)$, and let $\mathcal{E}_{c_a} = \langle \kappa, \mathbf{f} \rangle$ be the EVMDD corresponding to c_a. State s^{DD} determines a unique path through the EVMDD, which uniquely corresponds to an action sequence $a^0, \ldots, a^{k+1}$ (for some $k \in \{0, \ldots, level(\mathbf{f})\}$) with $a^0 = a^{\mathrm{pre}}$ and $a^{k+1} = a^{\mathrm{eff}}$. By construction, $\sum_{j=0}^{k+1} c_{a^j}(s) = c_a(s)$. Furthermore, also by construction, a^{pre} is applicable in s^{DD} iff a is applicable in s, and a^{eff} applies effects such that s'^{DD} corresponds to s'.

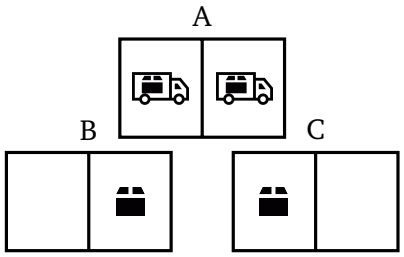
For necessity we have to show that there is no other transition sequence from s to s' in $\mathcal{T}(\Pi^{\mathrm{DD}})$. For this it suffices to see that semaphore variable lock prohibits "interleaving" more than one EVMDD evaluation at the same time, and that each aux^a makes sure that the corresponding EVMDD is traversed in the unique correct order. More formally, in every state s^{DD} of $\mathcal{T}(\Pi^{\mathrm{DD}})$ with $s^{\mathrm{DD}}(\mathrm{lock}) = 1$, only a unique action is applicable, and the next decision to be made only occurs when the semaphore is reset to zero by a concluding action. □

Corollary 2. *Let Π be a planning task, then:*

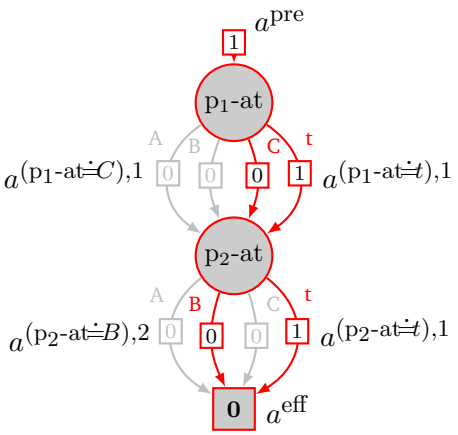
1. *Π^{DD} is a classical planning task,*
2. *$|\mathcal{A}^{\mathrm{DD}}|$ is in the order of $\max_{a \in \mathcal{A}} |\mathcal{E}_{c_a}|$,*
3. *There exists a plan π for Π, if and only if there exists a plan π' for Π^{DD} such that for each action a in π there is a sequence of actions in π' which is uniquely determined by the state where a is applied, and the cost of this sequence of actions is equal to the cost of a applied in this state. Therefore, $cost(\pi) = cost(\pi')$.*

Proof. The first two follow from construction of Π^{DD}. The last statement follows from Lemma 5 and that the goal condition of Π^{DD} ensures that we cannot pad a plan with irrelevant partial starting actions to obtain another different plan. □

Thus, we now have a compilation of tasks with state-dependent action costs, which is *polynomial in the size of the cost function representation*. If the EVMDD representations of the cost functions are small, then we have an efficient compilation *which also preserves plans* (modulo intermediate action sequences introduced by the compilation) *and plan cost*. Furthermore, this compilation has an additional benefit which will be of advantage when we consider different types of heuristics: if we have a Cartesian set of states



(a) Cartesian set of states s_C.



(b) EVMDD of the drive-AB cost function. Paths which are consistent with s_C are marked red.

Figure 2.13

s_C and an action a applicable in states of s_C, then we will have an action sequence applicable in the corresponding Cartesian set of compiled states $\text{DD}_{\mathcal{S}}(s_C) = \{s^{\text{DD}} | s \in s_C\}$, which is induced by a path in the EVMDD of a which is consistent with s_C. Figure 2.13 depicts the underlying idea for this. Once again, the Cartesian set of states $s_C = \{A\} \times \{C, t\} \times \{B, t\}$ implies four consistent paths in the EVMDD of the cost function representation. Every path then induces a sequence of actions, starting with the initial action a^{pre}, then either contain $a^{(\text{p}_1\text{-at}\doteq C),1}$ or $a^{(\text{p}_1\text{-at}\doteq t),1}$ and then $a^{(\text{p}_2\text{-at}\doteq B),2}$ or $a^{(\text{p}_2\text{-at}\doteq t),2}$, and a^{eff}. The cheapest cost of these sequences of actions is 1, which corresponds to the minimum cost of applying the drive action in any $s \in s_C$.

Lemma 6. *Let a be an action in Π and let $\mathcal{E}_{c_a} = \langle \kappa, \mathbf{f} \rangle$ be a corresponding EVMDD representation. Let s_C be a Cartesian set of states, let $s \in s_C$ and let $a \in \mathcal{A}$ be an action applicable in s. Let P^{s_C} be the set of paths from $\mathbf{f}$ to $\mathbf{0}$ consistent with s_C, i.e. $p \in P^{s_C}$ if $s_p \subseteq s'$ for some $s' \in s_C$. Then, for each $p \in P^{s_C}$ there is a sequence of actions $\pi = \langle a^{\text{pre}}, a_n, \ldots, a_1, a^{\text{eff}} \rangle$, such that π is applicable in s^{DD} and $s^{\text{DD}}[\pi]$ coincides with $s[a]$ in all facts not introduced by compilation. Furthermore, $\min_{s' \in s_C} c_a(s') \leq cost(\pi) = cost(p)$. Additionally, there exists $p \in P^{s_C}$ such that $\min_{s' \in s_C} c_a(s') = cost(\pi)$.*

Proof. A path $p = \langle \mathbf{v}_n, d_{v_n}, \ldots, \mathbf{v}_1, d_{v_1}, \mathbf{v}_0 \rangle$ with $\mathbf{v}_n = \mathbf{f}$ and $\mathbf{v}_0 = \mathbf{0}$ implies a sequence of actions $\pi = \langle a^{\text{pre}}, a_n, \ldots, a_1, a^{\text{eff}} \rangle$, where $a_i = a^{(\text{var}(\mathbf{v}_i) \doteq d_i), idx(\mathbf{v}_i)}$ $= \langle (\text{aux}^a \doteq idx(\mathbf{v}_i) \wedge (\text{var}(\mathbf{v}_i) \doteq d_i), (\text{aux}^a \doteq idx(\mathbf{v}_{i-1}) \rangle$, i.e. the action induced by the EVMDD compilation corresponding to this variable assignment,

for $i \in \{1, \ldots, n\}$. Now, π is applicable in s^{DD}, since, by definition, a^{pre} is applicable in s^{DD}, and each a_i is applicable after $\langle a^{\mathrm{pre}}, \ldots\ a_{i+1}\rangle$ was applied in s^{DD}, since p is consistent with s. Thus, a^{eff} is also applicable and its effects coincide with a in all variables not introduced by compilation. Then, $s^{\mathrm{DD}}[\pi]$ also coincides with $s[a]$. By construction, we have $\min_{s' \in s_C} c_a(s') \leq cost(\pi)$, since p corresponds to s and the sum of the weights is exactly $c_a(s)$. Furthermore, there has to be a path $p^\star$ corresponding to the state which minimizes the action cost function, and then $cost(\pi) = \min_{s' \in s_C} c_a(s')$. □

This result will be important when we consider the behaviour of the EVMDD compilation for heuristics which are based on Cartesian sets of states. There is one caveat though: the compilation introduces $|\mathcal{A}|+1$ additional variables, as we have one auxiliary variable aux^a for each action $a \in \mathcal{A}$ and additionally the semaphore variable lock. This might not sound much, but for some heuristics the number of auxiliary variables plays an important role. We therefore introduce a second type of EVMDD compilation, which results in nearly the same planning task, but only requires two additional variables. The idea is that instead of having separate auxiliary variables aux^a for each action a, we have a single variable aux and the domain of this variable is $\mathcal{D}_{\mathrm{aux}} = \{0, \ldots, \sum_{a \in \mathcal{A}} |\mathcal{D}_{\mathrm{aux}^a}|\}$. Informally, instead of resetting the numbering of EVMDD nodes between actions we continue the numbering. This can be seen as a bijective function which numbers the nodes of *all* EVMDDs.

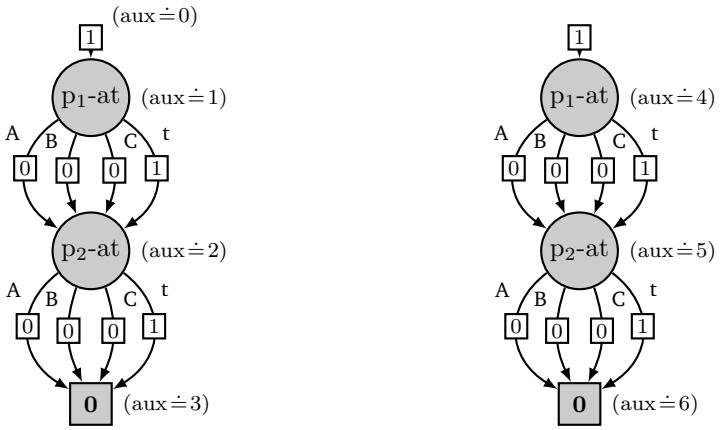
Definition 28 (Variable-compact EVMDD-based action compilation). Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ and $\mathcal{E}_i$ the EVMDD corresponding to the action cost function of a_i. By abuse of notation we define an index idx on each $\mathcal{E}$ such that $idx(\mathcal{E}_0) = 0$ and $idx(\mathcal{E}_{i+1}) = idx(\mathcal{E}_i) + |\mathcal{E}_i|$ for $i \in \{0, \ldots, n-1\}$.

Let $a = \langle \mathrm{pre}, \mathrm{eff}\rangle$ and aux be a variable with $\mathcal{D}_{\mathrm{aux}} = \{0, \ldots, idx(\mathcal{E}_n)+|\mathcal{E}_n|\}$. For each $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ of $\mathcal{E} = \mathcal{E}_{c_a}$ and each $i = 0, \ldots, k$ we get an action

$$a^{(v \doteq i), idx(\mathcal{E})+idx(\mathbf{v})} =$$
$$\langle (\mathrm{aux} \doteq idx(\mathcal{E}) + idx(\mathbf{v})) \wedge (v \doteq i), (\mathrm{aux} \doteq idx(\mathcal{E}) + idx(\chi_i(\mathbf{v})))\rangle$$

with constant cost w_i. As before, we denote the set of actions induced by $\mathbf{v}$ as $a^{\mathbf{v}}_{\mathcal{E}}$ and the set of actions induced by $\mathcal{E}$ as $a^{\mathcal{E}}$ (and they are defined analogous to Definition 25). Which definition we use will be clear from the context.

Example 10. Consider the logistics task and actions $\mathrm{drive\text{-}AB}$ and $\mathrm{drive\text{-}BA}$. Both actions have the same cost function. However, this action compilation assigns different auxiliary domain values to the nodes. Figure 2.14a depicts the EVMDD of $\mathrm{drive\text{-}AB}$. Nodes are annotated with the corresponding auxiliary domain values. Figure 2.14b depicts the EVMDD of $\mathrm{drive\text{-}BA}$. The EVMDDs have the same structure, but the index function idx assigns different domain values to these nodes. Intuitively, the index function numbers the EVMDD nodes continuously throughout all EVMDDs.



(a) EVMDD for cost function of drive-AB. (b) EVMDD for cost function of drive-BA.

Figure 2.14

Definition 29 (Variable-compact EVMDD compilation). Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_I, s_\star, c \rangle$ be a planning task. The *variable-compact EVMDD compilation* of Π is defined in the same way as Π^{DD}, but actions are compiled with Definition 28.

As already mentioned, variable-compact EVMDD compilation only introduces two additional variables. In all other regards it behaves the same as the original EVMDD compilation, but the original compilation is sometimes easier to depict as we can treat auxiliary values separately. Therefore, all results in this thesis will also be applicable to variable-compact EVMDD compilation[4]. Which compilation we use will be clear from the context.

It might not come to a surprise that we can also define a compilation which is very similar to the original EVMDD compilation, but is based on flattened EVMDDs instead of EVMDDs. Each node generates an action and its precondition and effect is defined exactly as in Definition 25. However, all actions induced by non-terminal nodes have a cost of 0. We also have an initial action with cost 0, and for each terminal node $\mathbf{w} = w$ we have a concluding action $a^{\mathbf{w}}$ with cost w.

Definition 30 (Flattened EVMDD-based action compilation). Let $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ be an action with cost function c_a, lock a variable and let $\mathcal{E}^{\mathcal{F}}_{c_a} = \mathbf{f}$ be a corresponding flattened EVMDD representation. Let $idx : \mathbf{v} \to \{1, \ldots, |\mathcal{E}_{c_a}|\}$ be a topological ordering of $\mathcal{E}^{\mathcal{F}}_{c_a}$, i.e. a bijective function numbering the nodes

[4]We do point that out here and do not repeat this in each proof about EVMDD compilation. It will be clear from the proof that a result for EVMDD compilation also holds for variable-compact EVMDD compilation, if requirements on the auxiliary variables are modified accordingly.

of $\mathcal{E}^{\mathcal{F}}_{c_a}$ such that for every non-terminal node $\mathbf{v}$ and each child $\chi_i(\mathbf{v}) : idx(\mathbf{v}) < idx(\chi_i(\mathbf{v}))$, and for two nodes $\mathbf{w}_i$, $\mathbf{w}_j$ we have $idx(\mathbf{w}_i) \neq \mathbf{w}_j$. Then, each non-terminal node $\mathbf{v}$ of $\mathcal{E}^{\mathcal{F}}_{c_a}$ induces a set of actions $a^{\mathbf{v}}$, defined as in Definition 25 but with a cost of 0. Additionally, each terminal node $\mathbf{w} = w$ of $\mathcal{E}^{\mathcal{F}}_{c_a}$ induces an action $a^{\mathbf{w}} = \langle(\mathrm{aux}^a \doteq idx(\mathbf{w})), (\mathrm{aux}^a \doteq 0) \wedge (\mathrm{lock} \doteq 0) \wedge \mathrm{eff}\rangle$ with cost w. Finally, we have an initial action $a^{\mathrm{pre}} = \langle \mathrm{pre} \wedge (\mathrm{lock} \doteq 0) \wedge (\mathrm{aux}^a \doteq 0), (\mathrm{lock} \doteq 1) \wedge (\mathrm{aux}^a \doteq 1)\rangle$ with cost 0. We denote the set of actions induced by $\mathcal{E}^{\mathcal{F}}_{c_a}$ as $a^{\mathcal{E}^{\mathcal{F}}_{c_a}} = \{a^{\mathrm{pre}}\} \cup \bigcup_{\mathbf{v}\in\mathcal{I}} a^{\mathbf{v}}$, where $\mathcal{I}$ denotes the set of non-terminal nodes in $\mathcal{E}^{\mathcal{F}}_{c_a}$.

Example 11. Consider the EVMDD compilation of action $\mathrm{drive\text{-}AB} = \langle(\text{t-at} \doteq A), (\text{t-at} \doteq B)\rangle$ from our logistics task. The flattened EVMDD of $c_{\mathrm{drive\text{-}AB}}$ is depicted in Figure 2.11. The domain of the auxiliary variable $\mathrm{aux}^{\mathrm{drive\text{-}AB}}$ is $\{0, \ldots, 6\}$. Again, we write a for drive-AB and aux for $\mathrm{aux}^{\mathrm{drive\text{-}AB}}$, and we get the following actions:

$$
\begin{aligned}
a^{\mathrm{pre}} &= \langle(\text{t-at} \doteq A) \wedge (\mathrm{lock} \doteq 0) \wedge (\mathrm{aux} \doteq 0), (\mathrm{lock} \doteq 1) \wedge (\mathrm{aux} \doteq 1)\rangle \\
a^{(\text{p}_1\text{-at}\doteq A),1} &= \langle(\mathrm{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq A), (\mathrm{aux} \doteq 2)\rangle \\
a^{(\text{p}_1\text{-at}\doteq B),1} &= \langle(\mathrm{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq B), (\mathrm{aux} \doteq 2)\rangle \\
a^{(\text{p}_1\text{-at}\doteq C),1} &= \langle(\mathrm{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq C), (\mathrm{aux} \doteq 2)\rangle \\
a^{(\text{p}_1\text{-at}\doteq t),1} &= \langle(\mathrm{aux} \doteq 1) \wedge (\text{p}_1\text{-at} \doteq t), (\mathrm{aux} \doteq 3)\rangle \\
a^{(\text{p}_2\text{-at}\doteq A),2} &= \langle(\mathrm{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq A), (\mathrm{aux} \doteq 4)\rangle \\
a^{(\text{p}_2\text{-at}\doteq B),2} &= \langle(\mathrm{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq B), (\mathrm{aux} \doteq 4)\rangle \\
a^{(\text{p}_2\text{-at}\doteq C),2} &= \langle(\mathrm{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq C), (\mathrm{aux} \doteq 4)\rangle \\
a^{(\text{p}_2\text{-at}\doteq t),2} &= \langle(\mathrm{aux} \doteq 2) \wedge (\text{p}_2\text{-at} \doteq t), (\mathrm{aux} \doteq 5)\rangle \\
a^{(\text{p}_2\text{-at}\doteq A),3} &= \langle(\mathrm{aux} \doteq 3) \wedge (\text{p}_2\text{-at} \doteq A), (\mathrm{aux} \doteq 5)\rangle \\
a^{(\text{p}_2\text{-at}\doteq B),3} &= \langle(\mathrm{aux} \doteq 3) \wedge (\text{p}_2\text{-at} \doteq B), (\mathrm{aux} \doteq 5)\rangle \\
a^{(\text{p}_2\text{-at}\doteq C),3} &= \langle(\mathrm{aux} \doteq 3) \wedge (\text{p}_2\text{-at} \doteq C), (\mathrm{aux} \doteq 5)\rangle \\
a^{(\text{p}_2\text{-at}\doteq t),3} &= \langle(\mathrm{aux} \doteq 3) \wedge (\text{p}_2\text{-at} \doteq t), (\mathrm{aux} \doteq 6)\rangle \\
a^{\mathbf{1}} &= \langle(\mathrm{aux} \doteq 4), (\text{t-at} \doteq B) \wedge (\mathrm{lock} \doteq 0) \wedge (\mathrm{aux} \doteq 0)\rangle \\
a^{\mathbf{2}} &= \langle(\mathrm{aux} \doteq 5), (\text{t-at} \doteq B) \wedge (\mathrm{lock} \doteq 0) \wedge (\mathrm{aux} \doteq 0)\rangle \\
a^{\mathbf{3}} &= \langle(\mathrm{aux} \doteq 6), (\text{t-at} \doteq B) \wedge (\mathrm{lock} \doteq 0) \wedge (\mathrm{aux} \doteq 0)\rangle
\end{aligned}
$$

Actions $a^{\mathbf{1}}$, $a^{\mathbf{2}}$, $a^{\mathbf{3}}$ have a cost of 1,2,3, respectively; all other actions have a cost of 0.

Definition 31 (Flattened EVMDD compilation). Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_I, s_\star, c \rangle$ be a planning task. The *flattened EVMDD compilation* of Π, denoted as Π^{FDD}, is the compilation where

- $\mathrm{FDD}_{\mathcal{V}}(\Pi) = \mathrm{DD}_{\mathcal{V}}(\Pi)$,
- $\mathrm{FDD}_{\mathcal{A}}(\Pi) = \bigcup_{a \in \mathcal{A}} a^{\mathcal{E}^{\mathcal{F}}_{ca}}$,
- $\mathrm{FDD}_{\mathcal{S}}(s) = \mathrm{DD}_{\mathcal{S}}(s)$,
- $\mathrm{FDD}_{s_\star}(\Pi) = \mathrm{DD}_{s_\star}(\Pi)$,
- and costs as above.

For flattened EVMDD compilation we get the same results as for EVMDD compilation. As the proofs are analogous to the proofs for the EVMDD compilation, we won't repeat them here. Every path still corresponds to the cost of the action applied in a state, the only difference is that the path costs are now summed up in the concluding actions, instead of distributed over intermediate actions.

Corollary 3. *Let Π be a planning task, then:*

1. *Π^{FDD} is a classical planning task,*
2. *$|\mathcal{A}^{\mathrm{FDD}}|$ is in the order of $\max_{a \in \mathcal{A}} |\mathcal{E}^{\mathcal{F}}_{ca}|$,*
3. *There exists a plan π for Π, if and only if there exists a plan π' for Π^{FDD} such that for each action a in π there is a sequence of actions in π' which is uniquely determined by the state where a is applied, and the cost of this sequence of actions is equal to the cost of a applied in this state. Therefore, $cost(\pi) = cost(\pi')$.*

The results for Cartesian sets of states are also the same. We won't repeat them here, but we note that this compilation introduces a large amount of zero cost actions and, additionally, the cheapest sequence of actions can only be determined when all concluding actions were evaluated. Note also that we can define flattened variable-compact EVMDD compilation in the same way as variable-compact EVMDD compilation.

The remaining question is how these compilations perform when heuristics are involved. It is important to point out that when we consider how a heuristic performs under some compilation $\mathcal{C}$ we are only interested in the heuristic estimates, but not in the behaviour of the underlying search algorithm on the compiled task. Since we have a corresponding state in the compiled task for each state in the original task (by application of $\mathcal{C}_{\mathcal{S}}$) we can perform search on the original task, and when we are required to compute the heuristic estimate of a state s, we can compute the heuristic estimate of

$\mathcal{C}_{\mathcal{S}}(s)$. This way of separating search and heuristic computation has multiple advantages: first, sometimes the heuristic does not care about specific parts of the compilation, e.g. some heuristics do not require the semaphore variable in the EVMDD compilation. Since search is not affected by the compilation we can in such cases just leave out the semaphore. Next, even if the EVMDDs underlying the EVMDD compilation are compact in size compilation still increases the size of the state space, which may increase memory consumption as the $A^\star$-algorithm has to store already expanded states in the closed list. Finally, note that our definition of heuristic invariance only compares states for which we have a one to one correspondence (via $\mathcal{C}_{\mathcal{S}}$), but EVMDD compilation introduces many intermediate states. In his Master's thesis, Wacker (2017) investigates how search behaves on the original task in comparison to the compiled task, with a compilation invariant but inadmissible heuristic as guidance (thus, the plans are not optimal). He shows that the compilation often produces worse plans (in terms of costs), since intermediate states (and their heuristic estimates) affect search negatively, and that the search algorithm has to be adapted to avoid such pitfalls. We can circumvent these adaptations by employing search on the original task, and only perform compilation to compute heuristic estimates.[5]

With that in mind, the next chapter investigates a well known family of classical planning heuristics, presents a generalization of such heuristics to tasks with state-dependent action costs, and investigates how these heuristics behave in the compiled task.

[5]This was actually pointed out by Malte Helmert during the defense of Wacker (2017).

CHAPTER 3

Delete Relaxation Heuristics

The first family of heuristics that we investigate for planning tasks with state-dependent action costs is the family of delete relaxation heuristics (Bonet et al. 1997; McDermott 1996). Delete relaxation heuristics were proposed more than 20 years ago, when most planning tasks were modeled in the STRIPS formalism which separates *add effects*, effects which make propositions true, and *delete effects*, effects which make propositions false. A *delete-relaxed* version of a planning task is then identical to the original task, except that actions drop their delete effects. Optimal planning in the delete-relaxed task is NP-hard (Bylander 1994), but approximations like the additive heuristic h^{add} and the maximum heuristic h^{max} are usually fast to compute.

3.1 Delete-relaxed Planning Tasks

Since we assume that our planning tasks are defined over finite-domain state variables, we have to generalize the notion of delete relaxation to this setting. The idea is that state variables accumulate their values, rather than switching between them, and a relaxed state contains variables which may hold several values at once.[1]

Definition 32 (Relaxed states). A *relaxed partial variable assignment* is a function $s^+ : \mathcal{V} \to \bigcup_{v \in \mathcal{V}} 2^{\mathcal{D}_v}$ with $s^+(v) \subseteq \mathcal{D}_v$ (instead of $s(v) \in \mathcal{D}_v$). If s^+ assigns at least one value to each $v \in \mathcal{V}$, s^+ is called a *relaxed state*. The set of all relaxed states is called $\mathcal{S}^+$. Given an unrelaxed state $s \in \mathcal{S}$, we say that s^+ *subsumes* s, if and only if for all $v \in \mathcal{V}$, $s(v) \in s^+(v)$, and write $s \in s^+$ if s^+ subsumes s.

[1] "It is not entirely clear to whom the original formulation of monotonic relaxation for multi-valued variable domains should be attributed", see the footnote of Domshlak et al. (2015) on page 75.

In the relaxed setting, action preconditions and effects are still partial variable assignments, but their semantics change. An action is applicable if the relaxed state satisfies its precondition, and the relaxed successor state "adds" the effects to the current state. Formally, an action $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ is *relaxed applicable* in s^+ if and only if $\mathrm{pre}(v) \in s^+(v)$ where $\mathrm{pre}(v)$ is defined. If it is clear from the context we also just say a is applicable in s^+. Applying a in s^+ results in $s^{+\prime}$, with $s^{+\prime}(v) = s^+(v) \cup \{\mathrm{eff}(v)\}$ if $\mathrm{eff}(v)$ is defined, and $s^{+\prime}(v) = s^+(v)$ otherwise. Again, we write $s^+[a]$ for $s^{+\prime}$.

The remaining piece we require for our definition of relaxed tasks is the relaxed version of an action cost function. Since a relaxed state subsumes multiple concrete states, the question is what cost to associate with the relaxed state. The underlying idea of relaxation heuristics is to approximate the cost of a plan in the original task, and often we want to achieve admissible heuristics. Therefore, we define the action cost in a relaxed state to be the minimum cost of all states subsumed by the relaxed state.

Definition 33 (Relaxed action cost function). Let $c_a : \mathcal{S} \to \mathbb{Q}^+$ be an action cost function. Then c_a^+ is the *relaxed action cost function* with $c_a^+(s^+) = \min_{s \in s^+} c_a(s)$. Again, the set of actions $\mathcal{A}$ induces a *global relaxed cost function* $c^+ : \mathcal{A} \times \mathcal{S}^+ \to \mathbb{Q}^+$, such that $c^+(a, s^+) = c_a^+(s^+)$ for all $s^+ \in \mathcal{S}^+$.

We are now able to formalize the delete-relaxed version of a planning task.

Definition 34 (Delete relaxation). Let $\Pi = (\mathcal{V}, \mathcal{A}, s_I, s_\star, c)$ be a planning task. Then $\Pi^+ = (\mathcal{V}, \mathcal{A}, s_I^+, s_\star, c^+)$ is the *delete relaxation of* Π, where s_I^+ assigns to each variable $v \in \mathcal{V}$ the singleton set $\{s_I(v)\}$. The set of all states of Π^+ is denoted with $\mathcal{S}^+$.

Take our logistics task from Example 1. In the delete relaxation, whenever we load a package into the truck, in the subsequent relaxed state the location of the package takes on both values: it is located in the truck and at the same time located at its previous location. For example, Figure 3.1 depicts the relaxed state $s^+ = (\text{t-at} \doteq A) \wedge (\text{p}_1\text{-at} \doteq A) \wedge (\text{p}_1\text{-at} \doteq t) \wedge (\text{p}_2\text{-at} \doteq B)$, which is reached by applying action load-p_1 in the initial state. Here, package p_1 is simultaneously at location A, and in the truck.

For classical planning tasks, the delete relaxation is often much easier to solve. Finding a (not necessary optimal) plan in the delete relaxation of a classical planning task can be done in polynomial time. Finding an optimal plan is, however, still NP-hard (Bylander 1994). Nevertheless, the plan cost of an optimal plan in the relaxed setting can be used as an admissible heuristic for the original task at hand.

Example 12. Consider the logistics task from Example 1. Figure 3.2 depicts the transition system of the relaxed task. Unreachable states are omitted. Relaxed states are shown in a column-like way: the first column contains

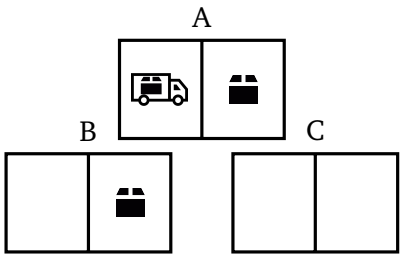


Figure 3.1: Relaxed state where the box is at position A and at the same time in the truck.

domain values of facts for $v = \text{t-at}$, the second one for $v = \text{p}_1\text{-at}$, and the third column stands for $v = \text{p}_2\text{-at}$. The initial state is marked in grey, a goal state is marked by double lines. Transitions induced by drive actions are colored red, transitions induced by load actions are colored blue, and transitions induced by unload actions are colored orange. Self-loops are omitted. Additionally, note that all actions have a cost of 1, since every relaxed state reachable from the initial state subsumes a state where the cost to drive is 1 (a state where both packages are at their start location). There are many optimal plans with a cost of 6.

Definition 35 (Optimal delete relaxation). Let $\Pi = (\mathcal{V}, \mathcal{A}, s_I, s_\star, c)$ be a planning task and $s \in \mathcal{S}$ a state. The *optimal delete relaxation heuristic* $h^+(s)$ is the cost of an optimal s^+-plan in Π^+, where s^+ assigns to each variable $v \in \mathcal{V}$ the singleton set $\{s(v)\}$

To be able to compute a delete relaxed optimal plan for a task with state-dependent action costs we have to include the notion of relaxed costs. The problem with this definition is not that much a conceptual, but rather a computational one, as there can be exponentially many unrelaxed states s subsumed by a single relaxed state s^+. However, observe that relaxed states are Cartesian sets of states.

Lemma 7 (Relaxed states are Cartesian sets of states). *A relaxed state s^+ is a Cartesian set of states.*

Proof. Follows from the definition of s^+. For each $v \in \mathcal{V}$ we have $s^+(v) \subseteq \mathcal{D}_v$, therefore we can write s^+ as $\times_{v \in \mathcal{V}} s^+(v)$, which is exactly the definition of a Cartesian set of states. □

Thus, when we have to compute the cost of a relaxed state we can make use of the local minimization introduced in Definition 19.

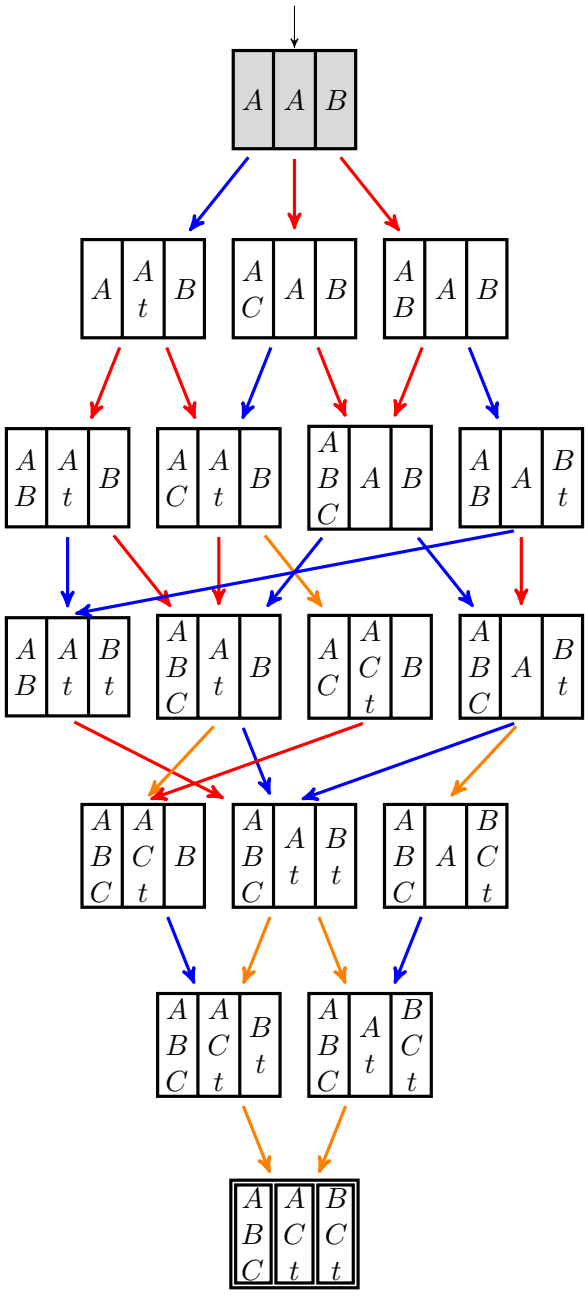


Figure 3.2: Transition system of the relaxed logistics task.

We now have a look into compilation approaches and investigate how these behave in the delete relaxed setting. Our first result shows that h^+ is invariant under exponential compilation.

Theorem 4. *Let Π be a planning task, Π^+ its delete relaxation, and Π^{EXP^+} the delete relaxation of its exponential compilation. Let s^+ be a state in Π^+.*

(a) If there exists an s^+-plan π for Π^+, then there exists an s^+-plan π^{EXP} for Π^{EXP^+}, with $cost(\pi) = cost(\pi^{\mathrm{EXP}})$.

(b) If there exists an s^+-plan π^{EXP} for Π^{EXP^+}, then there exists an s^+-plan π for Π^+ with $cost(\pi) \leq cost(\pi^{\mathrm{EXP}})$. If π^{EXP} is optimal, then $cost(\pi) = cost(\pi')$.

Proof. Since exponential compilation works on the same states as the original task, and since delete relaxation only changes the cost function of actions, we have $s^+ = \mathrm{EXP}_{\mathcal{S}}(s^+)$.

(a) Assume $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ is a plan for Π^+. Consider without loss of generality action $a = a_0$ applied in s^+. Let s be a state subsumed by s^+ such that $c_a(s) = c_a^+(s^+)$, i.e. a state subsumed by s^+ with minimal cost. By construction of the exponential compilation there is some action a', applicable in s, such that $c_{a'}(s) = c_a(s)$. Thus, a' is also applicable in s^+. Since this holds for all actions of π, we get a plan π' and $cost(\pi) = cost(\pi')$.

(b) Assume $\pi^{\mathrm{EXP}} = \langle a'_0, \ldots, a'_{n-1} \rangle$ is a plan for Π^{EXP^+}. Consider $a' = a'_0$, applied in s^+. Since actions (with state-dependent costs) in the original task have strictly fewer preconditions than actions in the compiled task, there is a corresponding action $a \in \mathcal{A}$, applicable in s^+. Since a' is applicable in s^+, the state s corresponding to the cost of a' is subsumed by s^+. Then $c_a^+(s^+) = \min_{s \in s^+} c_a(s) \leq c_{a'}^+$. If π' is optimal, then $\min_{s \in s^+} c_a(s) = c_{a'}^+$, because otherwise there would be an action a'' applicable in s^+ with cheaper cost.

□

Corollary 4. *The optimal delete-relaxation heuristic h^+ is invariant under exponential compilation.*

Proof. Follows from Theorem 4. Then, $h^+(s) = h^+(\mathrm{EXP}(s))$. □

As a consequence, instead of applying delete relaxation to the original task, we can use its exponential compilation to compute h^+. While we avoid the issue that a relaxed state subsumes in the worst case exponentially many

original states, we just shifted the computational overhead to the computation of the compilation. The obvious question is now, if we get a similar result for the EVMDD-based compilation. One particular property of this compilation is that by initiating a sequence of "intermediate" actions (based on the EVMDD structure), we make sure that no other chain of actions can be executed "concurrently". In the delete relaxed setting we can not enforce this sequential execution of a chain of actions, as every state reachable from the initial state contains the semaphore fact $(\mathrm{lock} \doteq 0)$. Therefore, one could assume that this compilation is not invariant under relaxation. Fortunately, it turns out that this is not the case. The key argument is that in the relaxed setting of the original task, after applying an action we can always choose the cheapest fact combination for the subsequent actions. This corresponds to concurrently evaluating intermediate actions in the compiled task.

Example 13. Consider Example 9, where we have two actions $a_x = \langle \top, (x \doteq 1) \rangle$ with $c_{a_x} = y$, $a_y = \langle \top, (y \doteq 1) \rangle$ with $c_{a_y} = x$, and state $s = (x \doteq 0) \wedge (y \doteq 0)$. In the relaxed setting, after applying a_x, the successor state contains facts $(x \doteq 0)$ *and* $(x \doteq 1)$, therefore application of a_y has a cost of 0. On the other hand, in the relaxation of the EVMDD compilation, the action sequence $\langle a_x^{\mathrm{pre}}, a_x^{(y \doteq 0),1}, a_y^{\mathrm{pre}}, a_y^{(x \doteq 0),1}, a_x^{\mathrm{eff}}, a_y^{\mathrm{eff}} \rangle$ is now allowed, since every successor state contains the condition $(\mathrm{lock} \doteq 0)$. Furthermore, since in actions in the delete relaxation preserve previous facts, the sequence $\langle a_x^{\mathrm{pre}}, a_x^{(y \doteq 0),1}, a_x^{\mathrm{eff}}, a_y^{\mathrm{pre}}, a_y^{(x \doteq 0),1}, a_y^{\mathrm{eff}} \rangle$ is also allowed.

Example 13 exhibits an important property of the EVMDD compilation: the semaphore variable is not necessary in the delete relaxed setting. For the remainder of this section, we assume that Π is a planning task, Π^+ its delete relaxation, and Π^{DD^+} the delete relaxation of its EVMDD compilation. Given a relaxed state s^+ of Π^+, we denote with $s^{+^{\mathrm{DD}}}$ the state $s^+ \cup \{(\mathrm{aux}^a \doteq 0) | a \in \mathcal{A}\}$, where $\mathcal{A}$ are actions in Π.

Lemma 8. *Let s_I be the initial state of Π. All states s^+ in Π^{DD^+} which are reachable from $\mathrm{DD}(s_I)$ contain the fact $(\mathrm{lock} \doteq 0)$.*

Proof. By definition of the EVMDD compilation we have $\mathrm{DD}(s_I) = s_I \wedge (\mathrm{lock} \doteq 0) \wedge \bigwedge_{a \in \mathcal{A}} (\mathrm{aux}^a \doteq 0)$. Actions in the delete relaxation only add facts to relaxed states, therefore the statement holds. □

As a consequence, when we are only interested in delete relaxation heuristics, we do not have to include the semaphore variable in the compilation.

Since relaxed states are Cartesian sets, we know from Lemma 6 that every path in the EVMDD induces a sequence of actions in the EVMDD compiled task. Furthermore, while in the delete relaxed setting the auxiliary values are not reset after applying such a sequence, we could in theory immediately

apply the concluding action again, without going through another chain of intermediate nodes. However, this would imply that we want to achieve the effect of an action a second time, which is not necessary in the delete relaxed setting[2]. With this in mind, we now give the proof that h^+ is invariant under compilation. For this, we show that an optimal plan in the relaxed original task has the same cost as an optimal plan in the relaxation of the compiled problem.

Theorem 5. *Let Π be a planning task, Π^+ its delete relaxation, and Π^{DD^+} the delete relaxation of its EVMDD compilation. Let s^+ be a state in Π^+ and $s^{+\prime} = s^{+\mathrm{DD}}$.*

(a) If there exists an optimal s^+-plan π for Π^+, then there exists an optimal $s^{+\prime}$-plan π' for Π^{DD^+}, with $cost(\pi) = cost(\pi')$.

(b) If there exists an optimal $s^{+\prime}$-plan π' for Π^{DD^+}, then there exists an optimal s^+-plan π for Π^+ with $cost(\pi) = cost(\pi')$.

Proof. (a) Assume $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ is an optimal s^+-plan for Π^+. We prove by induction over the plan length that for each a_i, $i \in \{0, \ldots, n-1\}$, there has to be a cheapest sequence of actions π^{a_i} in Π^{DD^+} with $c_{a_i}(s^+) = cost(\pi^{a_i})$, such that a_{i+1} is applicable in $s^{+\mathrm{DD}}_{i+1} = s^{+\mathrm{DD}}[\pi^{a_0}] \ldots [\pi^{a_i}]$ if $i \neq n-1$. Then, the sequence $\langle \pi^{a_0}, \ldots \pi^{a_{n-1}} \rangle$ is an optimal plan for $s^{+\mathrm{DD}}$ with cost $cost(\pi)$.

For the base case a_0, Lemma 6 implies that there exists a sequence of actions π^{a_0}, such that $c_{a_0}(s^+) = cost(\pi^{a_0})$ and $s^{+\prime}[\pi^{a_0}]$ coincides with $s^+[a_0]$ in all facts not introduced by the compilation. Now, assume the induction hypothesis holds for $a_0, \ldots, a_{i-1}$ and consider action a_i and state $s^+{}_i = s^+[a_0] \ldots [a_{i-1}]$. By induction hypothesis, a_i is applicable in $s^{+\mathrm{DD}}_i$. We can not immediately apply Lemma 6, as $s^{+\mathrm{DD}}_i$ contains additional facts $(\mathrm{aux}^{a_j} \doteq d)$ for some $j < i$ and $d \neq 0$, i.e. $s^+{}_i \neq s^{+\mathrm{DD}}_i$. However, these facts only affect actions in $\mathrm{DD}(a_j)$, and the effect of a_j is already achieved in $s^{+\mathrm{DD}}_i$. Since π is optimal, a_j is never applied twice, and we have $a_i \neq a_j$. Therefore, the action sequence π^{a_i} induced by Lemma 6 for $s^+{}_i$ is also the cheapest action sequence applicable in $s^{+\mathrm{DD}}_i$. If $i \neq n-1$, then a_{i+1} is applicable in $s^{+\mathrm{DD}}_{i+1}$, as the concluding action of π^{a_i} has the same effect as a_i (minus auxiliary fact).

(b) We now show the other direction by proving that given an optimal plan $\pi^{\mathrm{DD}} = \langle a_j^{\mathrm{pre}} \ldots, a_0^{\mathrm{eff}}, \ldots, a_{n-1}^{\mathrm{eff}} \rangle$, we can construct an action sequence

[2]Note that this is not the case if actions have conditional effects, since in this case it might be required to apply an action multiple times, to achieve different effects.

$\pi = \langle a_0, \ldots, a_{n-1} \rangle$ with $a_i^{\text{eff}} \in \text{DD}(a_i)$ for $i \in \{0, \ldots, n-1\}$, such that ${s^+}_i^{\text{DD}} = {s^+}^{\text{DD}}[a_j^{\text{pre}}][\ldots][a_{i-1}^{\text{eff}}]$ coincides with ${s^+}_i = s^+[a_0] \ldots [a_{i-1}]$ in all variables not introduced by compilation. We show this by induction over the number of concluding actions in π^{DD}. Since the last action of π^{DD} has to be a concluding action (otherwise π^{DD} is not optimal), we then have a plan for Π^+.

In the base case, there are no concluding actions in π^{DD}. Therefore ${s^+}^{\text{DD}}[\pi^{\text{DD}}]$ and ${s^+}^{\text{DD}}$ can only differ in variables introduced by the EVMDD compilation and then the empty plan $\pi = \langle\rangle$ is an s^+-plan. For the induction step, consider a_i^{eff}. Before a_i^{eff} is applicable, π^{DD} has to include actions $a_i^{\text{pre}}, a_i^0, \ldots, a_i^k$, where a_i^{pre} enables[3] a_i^0, a_i^k enables a_i^{eff}, and a_i^j enables a_i^{j+1} for $j \in \{0, \ldots, k-1\}$. To see this, note that ${s^+}^{\text{DD}}$, by definition, does not contain $(\text{aux}^{a_i} \doteq d)$ with $d \neq 0$, and for a_i^{eff} to be applicable, we have to set aux^{a_i} to the corresponding value via a chain of intermediate actions, beginning with a_i^{pre}. Thus, $\text{pre}(a_i^{\text{pre}}) \in {s^+}_i^{\text{DD}}$. Now consider a_i, such that $a_i^{\text{eff}} \in \text{DD}(a_i)$. By induction hypothesis, ${s^+}_i^{\text{DD}}$ and ${s^+}_i$ coincide in variables not introduced by the compilation. In particular, this means that $\text{pre}(a_i^{\text{pre}}) \in {s^+}_i$. Since $\text{pre}(a_i) \subset \text{pre}(a_i^{\text{pre}})$, a_i is also applicable in ${s^+}_i$. Furthermore, $\text{eff}(a_i)$ and $\text{eff}(a_i^{\text{eff}})$ coincide in non-auxiliary facts, therefore, ${s^+}_i[a_i]$ and ${s^+}_i^{\text{DD}}[a_i^{\text{eff}}]$ also coincide in non-auxiliary facts, concluding the proof by induction.

What is left is to show that $cost(\pi^{\text{DD}}) = cost(\pi)$ and that π is optimal. To see this, note that for each a_i in π, there is a corresponding (not necessarily sequential) action sequence $a_i^{\text{pre}}, a_i^0, \ldots, a_i^k, a_i^{\text{eff}}$ in π^{DD}. Since π^{DD} is optimal, and since compilation does not introduce actions with negative cost, each a_i^j is applied exactly once and enables an auxiliary fact. Thus, this sequence corresponds to a minimal path p in the EVMDD of c_{a_i}, and $cost(p)$ is exactly the cost of c_{a_i} applied in s_p, otherwise there would be a state s subsumed by ${s^+}_i$ with cheaper cost, and then this state would imply a path in the EVMDD leading to a cheaper sequence of actions, which is not possible, since π^{DD} is optimal. With the same argument π has to be optimal, concluding the proof.

□

Corollary 5. *The optimal delete-relaxation heuristic h^+ is invariant under EVMDD compilation.*

Proof. Follows from Theorem 5. Then, $h^+(s) = h^+(\text{DD}(s))$. □

[3] a_i enables a_j in s, iff. a_i is applicable in s, a_j not applicable in s, and a_j applicable in $s[a_i]$.

This is a promising result, since the EVMDD compilation is polynomial in the size of the EVMDD representation. Furthermore, when we separate search and heuristic computation, Lemma 8 allows us to remove the semaphore variable from the compiled task. For the remainder of this chapter we will assume that the EVMDD compilation does not include a semaphore variable.

While the compilation approach deals with the issue (assuming compact EVMDDs) that computing relaxed costs may depend on exponentially many unrelaxed states, the computation of h^+ in itself is still NP-hard. Therefore, h^+ is usually not an efficient heuristic and we have to rely on approximations. We will now investigate how two of these approximations behave on tasks with state-dependent action costs.

3.2 Approximative Delete Relaxation Heuristics

Over the years, there have been different proposals for approximations to h^+. The *additive heuristic* h^{add} (Bonet et al. 1997) is a polynomial approximation which assumes that subgoals are independent (achieved with no side-effects). It is inadmissible, but usually quite fast to compute. The *maximum heuristic* h^{max} (Bonet and Geffner 1999) is very similar to h^{add}, since it only replaces a sum operation during the computation of h^{add} with the $\max$ operator. It is admissible, but usually less informed. Both heuristics are equivalent to h^+ when all actions have a single precondition and the goal depends on a single fact (Keyder and Geffner 2008). Unlike h^{add} and h^{max}, the *FF heuristic* h^{FF} (Hoffmann and Nebel 2001) makes no independence assumption and instead computes a (not necessary optimal) plan for the relaxed task, which is possible in polynomial time for unit cost tasks. This heuristic is still a strong heuristic for satisficing (i.e. non-optimal) planning, but it ignores (constant) action costs. The set-additive heuristic h^{sa} (Keyder and Geffner 2008) is a variation of h^{add} which is cost sensitive. It collects the "best support" of propositions on the plan path, which is a set of actions encoding the "best" actions to achieve these propositions. Finally, the cost-sharing heuristic h^{cs} (Mirkis and Domshlak 2007), motivated by work on heuristic search for probabilistic reasoning, generalizes the additive, the maximum and the FF heuristic by building relaxed planning graphs up to an upper bound of the plan length and propagating cost vectors through the graph.

While relaxation heuristics are still a powerful tool for satisficing planning, for optimal planning they are either not admissible (and therefore not suitable for $A^\star$) or are outperformed by other classes of heuristics. We therefore only consider two types delete relaxation heuristics, which both have the property of having a simple mathematical formulation, as well as being inherently able to handle non-uniform action costs: the additive heuristic and the maximum heuristic. While the former is an inadmissible heuristic and therefore not useful for optimal planning, it can still be used to better guide non-satisficing

planners for tasks with state-dependent action costs. We investigate how these two heuristics interact with the state-dependent action cost setting and move then on to a more powerful class of heuristics. We start with the definition of h^{add}. For this, we first require the notion of fact achievers.

Definition 36 (Achievers). Let f be a fact. We denote with $A(f)$ the set of *achievers* of f, i.e. the set of actions $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ with $f \in \mathrm{eff}$.

Definition 37 (Additive heuristic). Let Π be a *classical* planning task. The *additive heuristic* h^{add} is defined as follows:

$$h^{add}(s) = h^{add}_s(s_\star) \tag{3.1}$$

$$h^{add}_s(s_\mathrm{p}) = \sum_{f \in s_\mathrm{p}} h^{add}_s(f) \quad \text{and} \tag{3.2}$$

$$h^{add}_s(f) = \begin{cases} 0 & \text{if } f \in s \\ \min_{a \in A(f)} \left(h^{add}_s(\mathrm{pre}(a)) + c_a \right) & \text{otherwise,} \end{cases} \tag{3.3}$$

where s_p stands for a partial state and f for a fact.

The underlying idea of the additive heuristic is that it sums up the cost of all achievers of goal facts, and recursively does this summation for the preconditions of achievers of these facts. If facts in the goal are not independent, then this is an overestimation of the cost, because a single action may achieve multiple facts. Therefore, h^{add} is not admissible. It is a delete relaxation heuristic since it assumes that actions always add facts, but never delete facts. In the case of zero-cost actions, to ensure that h^{add} (and as we will see therefore also h^{max}) is well-defined, one has to initialize the h^{add} values with ∞. Then, the greatest fix point is always unique. See also the discussion in the footnote of Röger et al. 2014, p. 2.

For tasks with state-dependent action costs, the question is how to generalize Equation 3.3. Consider the following example.

Example 14. Consider a planning task with binary variables x, y, g and the following actions: $a_g = \langle \top, (g \doteq 1) \rangle$, with $c_{a_g} = 2x + 4y$, $a_{\neg x} = \langle \top, (x \doteq 0) \rangle$ with constant cost 4 and $a_{\neg y} = \langle \top, (y \doteq 0) \rangle$ with constant cost 1. Let $s = (x \doteq 1) \wedge (y \doteq 1) \wedge (g \doteq 0)$ and $s_\star = (g \doteq 1)$. To reach the goal, we have to apply a_g. If we apply a_g immediately, we can achieve $(g \doteq 1)$ with a cost of 6. However, if we first apply $a_{\neg y}$, we pay a cost of 1, but reduce the cost of applying a_g by 4. This allows us to reach the goal with a cost of 3, instead of 6.

Example 14 reveals that the generalization of the additive heuristic to state-dependent action costs should not only minimize over all achievers of a fact f, but also minimize over all possible situations where the achiever is applicable. We can see a similar behaviour of h^{add} when we analyze the heuristic under the exponential compilation.

Example 15. Consider Example 14 and the exponential action compilation $\mathrm{EXP}(a_g)$:

$$a_g^{(x\doteq 0)\wedge(y\doteq 0)} = \langle (x\doteq 0)\wedge(y\doteq 0), (g\doteq 1)\rangle,\ cost: 0$$
$$a_g^{(x\doteq 1)\wedge(y\doteq 0)} = \langle (x\doteq 1)\wedge(y\doteq 0), (g\doteq 1)\rangle,\ cost: 2$$
$$a_g^{(x\doteq 0)\wedge(y\doteq 1)} = \langle (x\doteq 0)\wedge(y\doteq 1), (g\doteq 1)\rangle,\ cost: 4$$
$$a_g^{(x\doteq 1)\wedge(y\doteq 1)} = \langle (x\doteq 1)\wedge(y\doteq 1), (g\doteq 1)\rangle,\ cost: 6$$

Then, we get the following heuristic values:

$$h_s^{add}((x\doteq 0)) = 4, \qquad h_s^{add}((x\doteq 1)) = 0,$$
$$h_s^{add}((y\doteq 0)) = 1, \qquad h_s^{add}((y\doteq 1)) = 0,$$

$$\begin{aligned} h_s^{add}((g\doteq 1)) = \quad & \min\{h_s^{add}((x\doteq 0)\wedge(y\doteq 0)) + 0, \\ & h_s^{add}((x\doteq 1)\wedge(y\doteq 0)) + 2, \\ & h_s^{add}((x\doteq 0)\wedge(y\doteq 1)) + 4, \\ & h_s^{add}((x\doteq 1)\wedge(y\doteq 1)) + 6\} \\ = \quad & \min\{4+1+0, 0+1+2, 4+0+4, 0+0+6\} \\ = \quad & 3 \end{aligned}$$

Basically, what happens in the example is that we minimize over the action cost *and the heuristic estimate we need to achieve this cost*. This observation is the basis for our formal definition of the additive heuristic for tasks with state-dependent action cost.

Definition 38 (Generalized additive heuristic). Let Π be a planning task. Recall that $P(c_a)$ is the set of consistent partial variable assignments over the support of c_a. The *additive heuristic* h^{add} is defined as in Definition 37, but Equation 3.3 is replaced by the base case below:

$$h_s^{add}(f) = \begin{cases} 0 & \text{if } f \in s \\ \min_{a\in A(f)} \min_{p\in P(c_a)} \left(h_s^{add}(\mathrm{pre}(a)\cup p) + c_a(p) \right) & \text{otherwise.} \end{cases} \tag{3′}$$

Note that since the estimation for a partial state is the sum of the estimations of the individual facts, and since the action precondition and the support of the action cost are disjoint, we can also write

$$\min_{a\in A(f)} \left(h_s^{add}(\mathrm{pre}(a)) + \min_{p\in P(c_a)} \left(h_s^{add}(p) + c_a(p) \right) \right).$$

We first show that the generalized version of h^{add} behaves like the classical version for actions with constant cost:

Lemma 9. *Let all actions in $\mathcal{A}$ have constant cost, and let s be a state and $f \notin s$ a fact. Then $h_s^{add}(f) = \min_{a \in A(f)} \left(h_s^{add}(\mathrm{pre}(a)) + c_a\right)$.*

Proof. We have $h_s^{add}(f) = \min_{a \in A(f)} \min_{p \in P(c_a)} \left(h_s^{add}(\mathrm{pre}(a) \cup p) + c_a(p)\right)$. Since a has constant cost, $P(c_a) = \{\emptyset\}$ and $h_s^{add}(\mathrm{pre}(a)) + c_a = \min_{p \in P(c_a)} \left(h_s^{add}(\mathrm{pre}(a) \cup p) + c_a(p)\right)$. □

We won't show the computation of the generalized version of h^{add} for Example 15, as it is pretty similar to the computation under the exponential compilation. This should not come as a surprise, as the exponential compilation is the inspiration of the generalization of the additive heuristic. As a consequence, this heuristic is invariant under the exponential compilation.

Theorem 6. *The generalized additive heuristic h^{add} is invariant under exponential compilation.*

Proof. By construction of h^{add} and Lemma 9. We denote with $A(f) \subseteq \mathcal{A}$ achievers in Π and with $A^{\mathrm{EXP}}(f) \subseteq \mathrm{EXP}_{\mathcal{A}}(\Pi)$ achievers in Π^{EXP}. Let s be a state, $s' = \mathrm{EXP}(s)$, and a an action in Π, such that $a \in A(f)$ for some fact $f \notin s$. Then, $\mathrm{EXP}(a) = \{a^p | p \in P(c_a)\}$, where $a^p = \langle \mathrm{pre} \cup p, \mathrm{eff} \rangle$ with $c_{a^p} = c_a(p)$, and therefore also $a^p \in A^{\mathrm{EXP}}(f)$. Thus,

$$
\begin{aligned}
h_{s'}^{add}(f) &= \min_{a \in A^{\mathrm{EXP}}(f)} \left(h_s^{add}(\mathrm{pre}(a)) + c_a\right) \\
&= \min_{a \in A(f)} \left(\min_{p \in P(c_a)} \left(h_s^{add}(\mathrm{pre}(a) \cup p) + c_a(p)\right) \right) \\
&= h_s^{add}(f).
\end{aligned}
$$

□

While this result allows us to compute the additive heuristic for tasks with state-dependent action costs, the compilation still introduces exponential overhead. We therefore now show that h^{add} is also invariant under the EVMDD compilation. The idea behind the proof is that in order to compute the heuristic estimate of an auxiliary fact aux corresponding to a node in the EVMDD, we have to minimize over the achievers of this fact. The achievers are the actions corresponding to edges leading to that node; their cost is determined by the edge weight, and their precondition is the predecessor auxiliary fact, together with the fact enabling this edge. Figure 3.3 depicts the idea behind the proof, applied to Example 14. Depicted is the EVMDD of the cost

function of a_g. For each weight, there is a dashed edge leading to the corresponding fact enabling this edge, annotated with the h^{add}-value of this fact. Then, we can compute the h^{add}-value of an auxiliary fact, by minimizing over the edge weights plus the h^{add}-values of the corresponding facts. Intuitively, we minimize recursively along the EVMDD path, and in the end choose the path corresponding to the partial state which minimizes the heuristic estimate and the action cost of this partial state. We first show that the heuristic estimate of an auxiliary fact of a node $\mathbf{v}$ can be expressed as the minimum over the weights and heuristic estimates among the paths to $\mathbf{v}$.

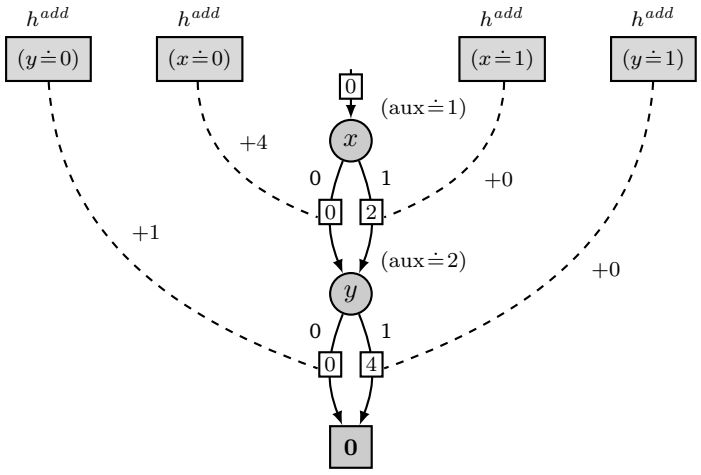


Figure 3.3: Proof sketch of Lemma 10 based on Example 14. At each decision node, the heuristic minimizes over the sum of the edge weight and the h^{add}-value of the corresponding fact. For example, the estimate of $(\mathrm{aux} \doteq 2)$ is the minimum of $0 + 4$ (for fact $(x \doteq 0)$) and $2 + 0$ (for fact $(x \doteq 1)$).

Lemma 10. *Let s be a state and $s^{\mathrm{DD}} = \mathrm{DD}(s)$ the corresponding state in the EVMDD compilation. Let $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ be an action with cost function c_a, aux^a the auxiliary variable introduced by the compilation, and let $\mathcal{E}_{c_a} = \langle \kappa, \mathbf{f} \rangle$ be the corresponding EVMDD representation and $\mathbf{v}$ some node of $\mathcal{E}_{c_a}$. We denote with $P(\mathbf{v})$ the set of paths from the root node to $\mathbf{v}$ and as before with $cost(p)$ the sum of weights along a path p and with s_p the partial variable assignment corresponding to p. Then*

$$h^{add}_{s^{\mathrm{DD}}}((\mathrm{aux}^a \doteq idx(\mathbf{v}))) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \min_{p \in P(\mathbf{v})} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right)$$

Proof. Proof by induction over the level of $\mathbf{v}$.

Base case $level(\mathbf{v}) = |\mathrm{vars}(c_a)|$: In the base case, $\mathbf{v} = \mathbf{f}$. Then $idx(\mathbf{v}) = 1$ and the single achiever of $(\mathrm{aux}^a \doteq 1)$ is $a^{\mathrm{pre}} = \langle (aux \doteq 0) \wedge \mathrm{pre}, (aux \doteq 1) \rangle$ with

cost κ. Therefore $h^{add}_{s^{\mathrm{DD}}}(\mathrm{aux}^a \doteq 1) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{aux}^a \doteq 0) + h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \kappa$. Since $(\mathrm{aux}^a \doteq 0) \in s^{\mathrm{DD}}$ we have $h^{add}_{s^{\mathrm{DD}}}(\mathrm{aux}^a \doteq 1) = 0 + h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \kappa$. Since $\mathbf{v}$ is the root node, there is a single path with weight κ, and the corresponding partial variable assignment is the empty set. Therefore

$$\begin{aligned} h^{add}_{s^{\mathrm{DD}}}((\mathrm{aux}^a \doteq idx(\mathbf{v}))) &= h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \kappa \\ &= h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + h^{add}_{s^{\mathrm{DD}}}(\emptyset) + \kappa \\ &= h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \min_{p \in P(\mathbf{v})} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right). \end{aligned}$$

Inductive step $level(\mathbf{v}) = i - 1$: Assume the claim is true for nodes $\mathbf{v}'$ with $level(\mathbf{v}') \geq i$. By definition of h^{add} and Lemma 9, we have

$$h^{add}_{s^{\mathrm{DD}}}((\mathrm{aux}^a \doteq idx(\mathbf{v})) = \min_{a' \in A((\mathrm{aux}^a \doteq idx(\mathbf{v}))} \left(h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}(a')) + c_{a'} \right).$$

Let $\mathbf{v}'$ be some parent of $\mathbf{v}$, $F_{\mathbf{v}'}$ the corresponding fact enabling the edge from $\mathbf{v}'$ to $\mathbf{v}$ and $w_{\mathbf{v}'}$ the weight of this edge. Then, there exists an achiever $a' = \langle (aux^a \doteq idx(\mathbf{v}')) \wedge F_{\mathbf{v}'}, (aux^a \doteq idx(\mathbf{v})) \rangle$ with $c_{a'} = w_{\mathbf{v}'}$. Therefore, $h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}(a')) = h^{add}_{s^{\mathrm{DD}}}((\mathrm{aux}^a \doteq idx(\mathbf{v}')) + h^{add}_{s^{\mathrm{DD}}}(F_{\mathbf{v}'})$. By induction hypothesis, the claim holds for node $\mathbf{v}'$ and therefore

$$h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}(a')) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \min_{p \in P(\mathbf{v}')} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right) + h^{add}_{s^{\mathrm{DD}}}(F_{\mathbf{v}'}).$$

Minimizing over all achievers of $(\mathrm{aux}^a \doteq idx(\mathbf{v}))$ corresponds to minimizing over all predecessors of $\mathbf{v}$, denoted as $Pr_{\mathbf{v}}$, and thus

$$\begin{aligned} &\min_{a' \in A((\mathrm{aux}^a \doteq idx(\mathbf{v}))} \left(h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}(a')) + c_{a'} \right) \\ &= \min_{\mathbf{v}' \in Pr_{\mathbf{v}}} \left(h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \min_{p \in P(\mathbf{v}')} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right) + h^{add}_{s^{\mathrm{DD}}}(F_{\mathbf{v}'}) + c_{a'} \right) \\ &= \min_{\mathbf{v}' \in Pr_{\mathbf{v}}} \left(h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \min_{p \in P(\mathbf{v}')} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right) + h^{add}_{s^{\mathrm{DD}}}(F_{\mathbf{v}'}) + w_{\mathbf{v}'} \right) \\ &= h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \min_{\mathbf{v}' \in Pr_{\mathbf{v}}} \left(\min_{p \in P(\mathbf{v}')} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right) + h^{add}_{s^{\mathrm{DD}}}(F_{\mathbf{v}'}) + w_{\mathbf{v}'} \right). \end{aligned}$$

Now, note that the inner minimization minimizes over the paths to the predecessors, and the outer minimization minimizes over the possible edges from the predecessor to $\mathbf{v}$. This is equal to minimizing over all possible paths to $\mathbf{v}$, and therefore

$$h^{add}_{s^{\mathrm{DD}}}(\mathrm{aux}^a \doteq idx(\mathbf{v})) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}) + \min_{p \in P(\mathbf{v})} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right).$$

□

With this, we are now able to show that h^{add} is invariant under EVMDD compilation, as long as the EVMDDs we use to represent cost functions are quasi-reduced. We require quasi-reducedness, since otherwise a path in the EVMDD might skip some facts and we lose their heuristic estimates.

Theorem 7. *The generalized additive heuristic h^{add} is invariant under EVMDD compilation, if the EVMDDs are quasi-reduced.*

Proof. Let Π be a planning task, s be a state and $s^{\mathrm{DD}} = \mathrm{DD}(s)$ the corresponding state in the compiled task. We have to show that $h^{add}(s) = h^{add}(s^{\mathrm{DD}})$. We have $h^{add}(s) = h^{add}_s(s_\star)$ and $h^{add}(s^{\mathrm{DD}}) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{DD}_{s_\star}(\Pi))$. Since $\mathrm{DD}_{s_\star}(\Pi) = s_\star \cup \{(\mathrm{aux}^a \doteq 0) | a \in \mathcal{A}\}$ and $s^{\mathrm{DD}}(\mathrm{aux}^a) = 0$ for all $a \in \mathcal{A}$ it suffices to show that $h^{add}_s(s_\star) = h^{add}_{s^{\mathrm{DD}}}(s_\star)$. Furthermore, since h^{add} for a set of facts is the sum of the estimates of the single facts we have to show $h^{add}_s(f) = h^{add}_{s^{\mathrm{DD}}}(f)$ for some fact f.

We do this by induction over the number of steps required to achieve f from s. In the base case, f is already achieved and we have $f \in s$, then $f \in s^{\mathrm{DD}}$ and $h^{add}_s(f) = 0 = h^{add}_{s^{\mathrm{DD}}}(f)$.

In the inductive case, the statement holds for all facts achieved from s in i steps, and f is achieved in $i+1$ steps. We have to minimize over the achievers of f. By definition of the EVMDD compilation, for each achiever $a \in A(f)$ in the original task there is an achiever $a^{\mathrm{eff}} \in A(f)$ in the compiled task with precondition $(\mathrm{aux}^a \doteq |\mathbf{f}|)$, $f \in \mathrm{eff}(a)$ and cost 0. We have to show that

$$h^{add}_s(\mathrm{pre}(a)) + \min_{p \in P(c_a)} \left(h^{add}_s(p) + c_a(p) \right) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{aux}^a \doteq |\mathbf{f}|).$$

Since $idx(\mathbf{0}) = |\mathbf{f}|$ we get from Lemma 10

$$h^{add}_{s^{\mathrm{DD}}}(\mathrm{aux}^a \doteq |\mathbf{f}|) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}(a)) + \min_{p \in P(\mathbf{0})} \left(h^{add}_s(s_p) + cost(p) \right).$$

Since the EVMDD is quasi-reduced, every path from the root node to the terminal node $\mathbf{0}$ corresponds to a partial state of $P(c_a)$. Furthermore, by construction of the EVMDD, the weights among the path from the root node to $\mathbf{0}$ correspond exactly to the cost of the corresponding partial state, therefore

$$\min_{p \in P(c_a)} \left(h^{add}_{s^{\mathrm{DD}}}(p) + c_a(p) \right) = \min_{p \in P(\mathbf{0})} \left(h^{add}_{s^{\mathrm{DD}}}(s_p) + cost(p) \right)$$

and we have

$$h^{add}_{s^{\mathrm{DD}}}(\mathrm{aux}^a \doteq |\mathbf{f}|) = h^{add}_{s^{\mathrm{DD}}}(\mathrm{pre}(a)) + \min_{p \in P(c_a)} \left(h^{add}_{s^{\mathrm{DD}}}(p) + c_a(p) \right).$$

Since $\mathrm{pre}(a)$ and p have to be achievable in $i-1$ steps (otherwise $\mathrm{pre}(a)$ would not be applicable, and $h^{add}_{s^{\mathrm{DD}}}(p)$ would be ∞), we get by induction hypothesis:

$$h_s^{add}(\mathrm{pre}(a)) + \min_{p \in P(c_a)} \left(h_s^{add}(p) + c_a(p) \right) =$$
$$h_{s^{\mathrm{DD}}}^{add}(\mathrm{pre}(a)) + \min_{p \in P(c_a)} \left(h_{s^{\mathrm{DD}}}^{add}(p) + c_a(p) \right).$$

With this, it follows $h^{add}(s) = h^{add}(s^{\mathrm{DD}})$, concluding the proof. □

3.2.1 The maximum heuristic h^{max}

While the additive heuristic accumulates the heuristic estimates of goal facts, the maximum heuristic h^{max} only accounts for the maximum cost over all achievers of goal facts. As a consequence, h^{max} is an admissible heuristic. Below we give the definition of the maximum heuristic for classical planning tasks.

Definition 39 (Maximum heuristic). Let Π be a *classical* planning task. The *maximum heuristic* h^{max} is defined as follows:

$$h^{max}(s) = h_s^{max}(s_\star) \tag{3.4}$$
$$h_s^{max}(s_\mathrm{p}) = \max_{f \in s_\mathrm{p}} h_s^{max}(f) \quad \text{and} \tag{3.5}$$
$$h_s^{max}(f) = \begin{cases} 0 & \text{if } f \in s \\ \min_{a \in A(f)} \left(h_s^{max}(\mathrm{pre}(a)) + c_a \right) & \text{otherwise,} \end{cases} \tag{3.6}$$

where s_p stands for a partial state and f for a fact.

The only difference between the maximum and the additive heuristic is the maximization in Equation 3.5. Since this maximization is also possible for tasks with state-dependent action costs, we can use Equation 3′ of the generalization of the additive heuristic and thus formally define the generalization of the maximum heuristic.

Definition 40 (Generalized maximum heuristic). Let Π be a planning task. The *maximum heuristic* h^{max} is defined as in Definition 39, but Equation 3.6 is replaced by the base case below:

$$h_s^{max}(f) = \begin{cases} 0 & \text{if } f \in s \\ \min_{a \in A(f)} \left(\min_{p \in P(c_a)} \left(h_s^{max}(\mathrm{pre}(a) \cup p) + c_a(p) \right) \right) & \text{otherwise.} \end{cases} \tag{3′}$$

Unlike in the additive case, we are not able to separate the heuristic estimate of the precondition from the estimates of the partial state facts, i.e. write

$$\min_{a \in A(f)} \left(\max\{ h_s^{max}(\mathrm{pre}(a)), \min_{p \in P(c_a)} (h_s^{max}(p) + c_a(p)) \} \right),$$

since $\max(a,b)+c \neq \max(a,b+c)$ for arbitrary $a,b,c \in \mathbb{Q}^+$. We will come back to this observation when we investigate how h^{max} behaves under EVMDD compilation.

As it was the case for the additive heuristic, h^{max} also behaves like its classical counterpart when only constant action costs are involved.

Lemma 11. *Let Π be a classical planning task, s a state, and $f \notin s$ a fact. Then $h^{max}_s(f) = \min_{a \in A(f)} (h^{max}_s(\mathrm{pre}(a)) + c_a)$.*

Proof. We have $h^{max}_s(f) = \min_{a \in A(f)} \min_{p \in P(c_a)} (h^{max}_s(\mathrm{pre}(a) \cup p) + c_a(p))$. Since a has constant cost, $P(c_a) = \{\emptyset\}$ and then

$$\min_{p \in P(c_a)} (h^{max}_s(\mathrm{pre}(a) \cup p) + c_a(p)) = h^{max}_s(\mathrm{pre}(a)) + c_a.$$

□

Similarly, h^{max} is also invariant under exponential compilation.

Theorem 8. *The generalized maximum heuristic h^{max} is invariant under exponential compilation.*

Proof. The proof is analogous to the proof of Theorem 6. □

For the additive heuristic, a key argument for its invariance under EVMDD compilation is that we can construct the estimate of an auxiliary fact of a node $\mathbf{v}$, by minimizing over estimates of auxiliary fact and enabling edge fact of predecessor nodes, and adding the corresponding edge weight. For h^{max}, this is not possible. In principle, the underlying issue is that $\max(a,b) + c \neq \max(a, b+c)$, and as a result large heuristic estimates for facts enabling edges in the upper parts (nodes with higher level) of the EVMDD can "absorb" the cost of the path leading to nodes in the lower part of the EVMDD.

Example 16. Consider the planning task of Example 15 again. As a reminder, we have the actions $a_g = \langle \top, (g \doteq 1) \rangle$, with $c_{a_g} = 2x + 4y$, $a_{\neg x} = \langle \top, (x \doteq 0) \rangle$ with constant cost 4 and $a_{\neg y} = \langle \top, (y \doteq 0) \rangle$ with constant cost 1, and we have state $s = (x \doteq 1) \wedge (y \doteq 1) \wedge (g \doteq 0)$ and goal $s_\star = (g \doteq 1)$. We write c_a for c_{a_g}.

Then, $h^{max}(s) = h^{max}_s((g \doteq 1)) = \min_{p \in P(c_a)} (h^{max}(p) + c_a(p))$. Each p consists of a partial state defined over x and y and we have

$$\begin{aligned} s^{\neg x \neg y} &= (x \doteq 0) \wedge (y \doteq 0), & h^{max}_s((x \doteq 0)) &= 4, \\ s^{\neg x y} &= (x \doteq 0) \wedge (y \doteq 1), & h^{max}_s((x \doteq 1)) &= 0, \\ s^{x \neg y} &= (x \doteq 1) \wedge (y \doteq 0), & h^{max}_s((y \doteq 0)) &= 1, \\ s^{x y} &= (x \doteq 1) \wedge (y \doteq 1). & h^{max}_s((y \doteq 1)) &= 0. \end{aligned}$$

Thus,

$$\begin{aligned} h_s^{max}((g \doteq 1)) = \quad & \min\{h^{max}(s^{\neg x \neg y}) + c_a(s^{\neg x \neg y}), \quad h^{max}(s^{\neg x y}) + c_a(s^{\neg x y}), \\ & \quad h^{max}(s^{x \neg y}) + c_a(s^{x \neg y}), \quad h^{max}(s^{xy}) + c_a(s^{xy})\} \\ = \quad & \min\{\max\{4,1\} + 0, \quad \max\{4,0\} + 4, \\ & \quad \max\{0,1\} + 2, \quad \max\{0,0\} + 6\} \\ = \quad & 3. \end{aligned}$$

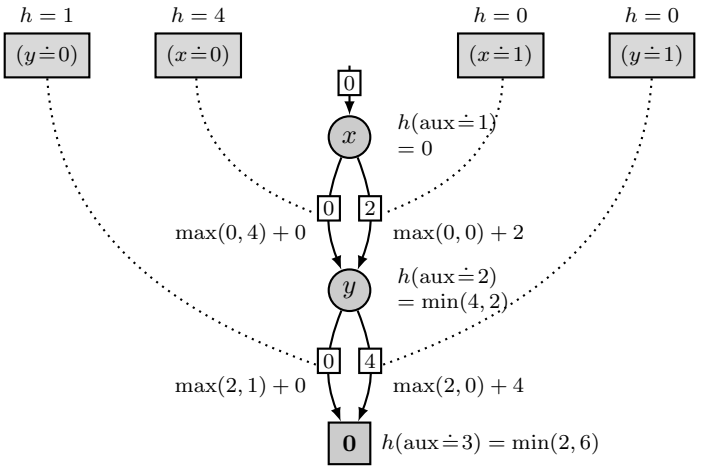


Figure 3.4: EVMDD for cost function $2x + 4y$, annotated with h^{max} values.

Now, consider the EVMDD compilation of a_g. Figure 3.4 depicts the EVMDD of c_a already annotated with the h-values of auxiliary facts which follow now. We denote aux^a with aux and have $s^{\mathrm{DD}} = s \wedge (aux \doteq 0)$. We get the following actions for $\mathrm{DD}(a)$:

$$\begin{aligned} a_g^{\mathrm{pre}} &= \langle (\mathrm{aux} \doteq 0), (\mathrm{aux} \doteq 1) \rangle,\ cost : 0 \\ a_g^{(x \doteq 0),1} &= \langle (\mathrm{aux} \doteq 1) \wedge (x \doteq 0), (\mathrm{aux} \doteq 2) \rangle,\ cost : 0 \\ a_g^{(x \doteq 1),1} &= \langle (\mathrm{aux} \doteq 1) \wedge (x \doteq 1), (\mathrm{aux} \doteq 2) \rangle,\ cost : 2 \\ a_g^{(y \doteq 0),2} &= \langle (\mathrm{aux} \doteq 2) \wedge (y \doteq 0), (\mathrm{aux} \doteq 3) \rangle,\ cost : 0 \\ a_g^{(y \doteq 1),2} &= \langle (\mathrm{aux} \doteq 2) \wedge (y \doteq 1), (\mathrm{aux} \doteq 3) \rangle,\ cost : 4 \\ a_g^{\mathrm{eff}} &= \langle (\mathrm{aux} \doteq 3), (g \doteq 1) \wedge (\mathrm{aux} \doteq 0) \rangle,\ cost : 0 \end{aligned}$$

We then get the following h^{max} values:

$$\begin{aligned} h^{max}_{s^{\mathrm{DD}}}((x \doteq 0)) &= 4, & h^{max}_{s^{\mathrm{DD}}}((x \doteq 1)) &= 0, \\ h^{max}_{s^{\mathrm{DD}}}((y \doteq 0)) &= 1, & h^{max}_{s^{\mathrm{DD}}}((y \doteq 1)) &= 0, \\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 0)) &= 0, & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 1)) &= 0, \end{aligned}$$

And for the auxiliary variables we have:

$$\begin{aligned} h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 2)) &= \min\{ & & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 1) \wedge (x \doteq 0)) + 0, \\ & & & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 1) \wedge (x \doteq 1)) + 2\} \\ &= \min\{ & & \max\{0, 4\} + 0, \max\{0, 0\} + 2\} \\ &= 2, \\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 3)) &= \min\{ & & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 2) \wedge (y \doteq 0)) + 0, \\ & & & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 2) \wedge (y \doteq 1)) + 4\} \\ &= \min\{ & & \max\{2, 1\} + 0, \max\{2, 0\} + 4\} \\ &= 2, \\ \Rightarrow h^{max}_{s^{\mathrm{DD}}}((g \doteq 1)) &= 2, \end{aligned}$$

whereas $h^{max}_{s}((g \doteq 1)) = 3$.

Theorem 9. *The generalized maximum heuristic h^{max} is not invariant under EVMDD compilation compilation.*

Proof. In Example 16, we have a state s with $h^{max}(s) \neq h^{max}(\mathrm{DD}(s))$. □

In principle, we could have used a simpler example, where the cost function only depends on a single variable, and some constant costs are "absorbed" by the heuristic. However, Example 16 also shows that it does not matter if we have the constant action cost as part of the concluding action, instead of the initial action.

Interestingly, this issue does not arise if we rely on flattened EVMDDs (Def. 20). Then, all non-concluding actions have a cost of 0 and the heuristic estimate of the effect minimizes over the estimates of the auxiliary values corresponding to the concluding actions and their cost.

Example 17. Consider Example 15 again, with the flattened EVMDD compilation of action a_g. Figure 3.5 depicts the flattened EVMDD of $c_{a_g} = 2x + 4y$. We denote aux^{a_g} with aux and have $s^{\mathrm{DD}} = s \wedge (aux \doteq 0)$, with $s = (x \doteq 1) \wedge (y \doteq 1) \wedge (g \doteq 0)$. We get the following actions for $\mathrm{FDD}(a_g)$:

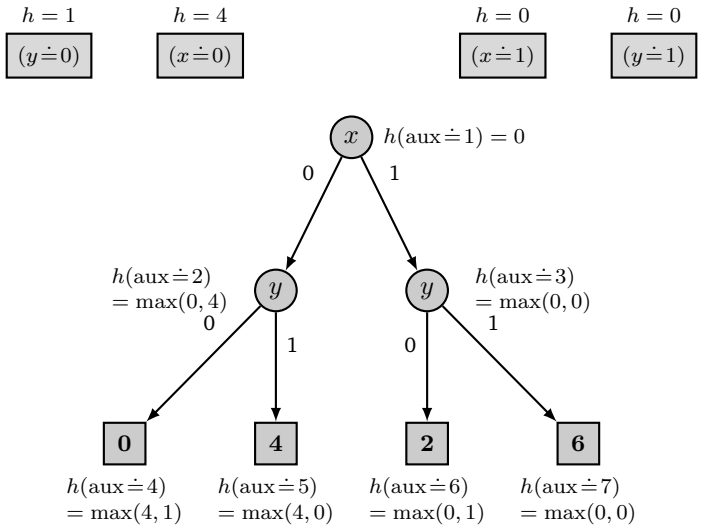


Figure 3.5: Flattened EVMDD for cost function $2x + 4y$, annotated with h^{max} values.

$$
\begin{aligned}
a_g^{\text{pre}} &= \langle (\text{aux} \doteq 0), (\text{aux} \doteq 1) \rangle,\ cost : 0 \\
a_g^{(x \doteq 0),1} &= \langle (\text{aux} \doteq 1) \wedge (x \doteq 0), (\text{aux} \doteq 2) \rangle,\ cost : 0 \\
a_g^{(x \doteq 1),1} &= \langle (\text{aux} \doteq 1) \wedge (x \doteq 1), (\text{aux} \doteq 3) \rangle,\ cost : 0 \\
a_g^{(y \doteq 0),2} &= \langle (\text{aux} \doteq 2) \wedge (y \doteq 0), (\text{aux} \doteq 4) \rangle,\ cost : 0 \\
a_g^{(y \doteq 1),2} &= \langle (\text{aux} \doteq 2) \wedge (y \doteq 1), (\text{aux} \doteq 5) \rangle,\ cost : 0 \\
a_g^{(y \doteq 0),3} &= \langle (\text{aux} \doteq 3) \wedge (y \doteq 0), (\text{aux} \doteq 6) \rangle,\ cost : 0 \\
a_g^{(y \doteq 1),3} &= \langle (\text{aux} \doteq 3) \wedge (y \doteq 1), (\text{aux} \doteq 7) \rangle,\ cost : 0 \\
a_g^{\mathbf{0}} &= \langle (\text{aux} \doteq 4), (g \doteq 1) \wedge (\text{aux} \doteq 0) \rangle,\ cost : 0 \\
a_g^{\mathbf{4}} &= \langle (\text{aux} \doteq 5), (g \doteq 1) \wedge (\text{aux} \doteq 0) \rangle,\ cost : 4 \\
a_g^{\mathbf{2}} &= \langle (\text{aux} \doteq 6), (g \doteq 1) \wedge (\text{aux} \doteq 0) \rangle,\ cost : 2 \\
a_g^{\mathbf{6}} &= \langle (\text{aux} \doteq 7), (g \doteq 1) \wedge (\text{aux} \doteq 0) \rangle,\ cost : 6
\end{aligned}
$$

We then get the following h^{max} values:

$$\begin{aligned} h^{max}_{s^{\mathrm{DD}}}((x \doteq 0)) &= 4, & h^{max}_{s^{\mathrm{DD}}}((x \doteq 1)) &= 0,\\ h^{max}_{s^{\mathrm{DD}}}((y \doteq 0)) &= 1, & h^{max}_{s^{\mathrm{DD}}}((y \doteq 1)) &= 0,\\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 0)) &= 0, & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 1)) &= 0, \end{aligned}$$

$$\begin{aligned} h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 2)) = \quad & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 1) \wedge (x \doteq 0)) + 0 = \max\{0, 4\} + 0 = 4,\\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 3)) = \quad & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 1) \wedge (x \doteq 1)) + 0 = \max\{0, 0\} + 0 = 0,\\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 4)) = \quad & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 2) \wedge (y \doteq 0)) + 0 = \max\{4, 1\} + 0 = 4,\\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 5)) = \quad & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 2) \wedge (y \doteq 1)) + 0 = \max\{4, 0\} + 0 = 4,\\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 6)) = \quad & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 3) \wedge (y \doteq 0)) + 0 = \max\{0, 1\} + 0 = 1,\\ h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 7)) = \quad & h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 3) \wedge (y \doteq 1)) + 0 = \max\{0, 0\} + 0 = 0,\\ h^{max}_{s^{\mathrm{DD}}}((g \doteq 1)) = \quad & \min\{h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 4)) + 0, h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 5)) + 4,\\ & \quad h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 6)) + 2, h^{max}_{s^{\mathrm{DD}}}((\mathrm{aux} \doteq 7)) + 6\}\\ = \quad & \min\{4 + 0, 4 + 4, 1 + 2, 0 + 6\} = 3 \end{aligned}$$

Note that the heuristic estimate in the compiled task corresponds to the heuristic estimate in the original task.

If we rely on flattened EVMDDs, we can give results similar to the additive heuristic.

Lemma 12. *Let s be a state and $s^{\mathrm{FDD}} = \mathrm{FDD}(s)$ the corresponding state in the flattened EVMDD compilation. Let $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$ be an action with cost function c_a, aux^a the auxiliary variable introduced by the compilation, and let $\mathcal{E}_{c_a} = \langle \kappa, \mathbf{f} \rangle$ be the corresponding EVMDD representation and $\mathbf{v}$ some node of $\mathcal{E}_{c_a}$. We denote with $P(\mathbf{v})$ the set of paths from the root node to $\mathbf{v}$ and with s_p the partial variable assignment corresponding to path p. Then*

$$h^{max}_{s^{\mathrm{FDD}}}(\mathrm{aux}^a \doteq idx(\mathbf{v})) = \min_{p \in P(\mathbf{v})} \left(h^{max}_{s^{\mathrm{FDD}}}(s_p \cup \mathrm{pre}) \right).$$

Proof. The proof by induction is similar to the proof of Lemma 10. Note that all non-concluding actions have a cost of 0. In the base case, we have $idx(\mathbf{v}) = 1$ and the single achiever is a^{pre} with cost 0. Thus, $h^{max}_{s^{\mathrm{FDD}}}(\mathrm{aux}^a \doteq 1) = h^{max}_{s^{\mathrm{FDD}}}(\mathrm{pre})$. Since s_p is the empty partial state, the claim holds.

In the induction step, assume the claim holds for nodes $\mathbf{v}'$ with higher level than $\mathbf{v}$. Let $\mathbf{v}'$ be some parent of $\mathbf{v}$ and $F_{\mathbf{v}'}$ the corresponding fact enabling edge. Then, there exists an achiever $a' = \langle (aux^a \doteq idx(\mathbf{v}')) \wedge F_{\mathbf{v}'}, (aux^a \doteq$

$idx(\mathbf{v}))\rangle$ with cost 0. Therefore,

$$\begin{aligned} h^{max}_{s^{\mathrm{FDD}}}(\mathrm{pre}(a')) &= h^{max}_{s^{\mathrm{FDD}}}((\mathrm{aux}^a \doteq idx(\mathbf{v}')) \cup F_{\mathbf{v}'}) \\ &= \max\{h^{max}_{s^{\mathrm{FDD}}}((\mathrm{aux}^a \doteq idx(\mathbf{v}'))), h^{max}_{s^{\mathrm{FDD}}}(F_{\mathbf{v}'})\}. \end{aligned}$$

By induction hypothesis, the claim holds for node $\mathbf{v}'$ and we have

$$\begin{aligned} h^{max}_{s^{\mathrm{FDD}}}(\mathrm{pre}(a')) &= \max\{\min_{p\in P(\mathbf{v}')}\left(h^{max}_{s^{\mathrm{FDD}}}(s_p \cup \mathrm{pre})\right), h^{max}_{s^{\mathrm{FDD}}}(F_{\mathbf{v}'})\} \\ &= \min_{p\in P(\mathbf{v}')}\left(\max\{h^{max}_{s^{\mathrm{FDD}}}(s_p \cup \mathrm{pre}), h^{max}_{s^{\mathrm{FDD}}}(F_{\mathbf{v}'}\}\right) \\ &= \min_{p\in P(\mathbf{v}')}\left(h^{max}_{s^{\mathrm{FDD}}}(s_p \cup \mathrm{pre} \cup F_{\mathbf{v}'})\right), \end{aligned}$$

where the last equality is obtained by definition of h^{max}. Again, since minimizing over all achievers of $(\mathrm{aux}^a \doteq idx(\mathbf{v}))$ corresponds to minimizing over all predecessors of $\mathbf{v}$, we get

$$\begin{aligned} &\min_{a'\in A(\mathrm{aux}^a \doteq idx(\mathbf{v}))} h^{max}_{s^{\mathrm{FDD}}}(\mathrm{pre}(a')) \\ &= \min_{\mathbf{v}'\in \text{pred. of } \mathbf{v}} \min_{p\in P(\mathbf{v}')} h^{max}_{s^{\mathrm{FDD}}}(\mathrm{pre} \cup s_p \cup F_{\mathbf{v}'}) \\ &= \min_{p\in P(\mathbf{v})} h^{max}_{s^{\mathrm{FDD}}}(s_p \cup \mathrm{pre}), \end{aligned}$$

concluding the proof. □

This brings us to the final theorem of this section, showing that h^{max} is invariant under quasi-reduced flattened EVMDD compilation.

Theorem 10. *The generalized maximum heuristic h^{max} is invariant under flattened EVMDD compilation, if the EVMDDs are quasi-reduced.*

Proof. Let Π be a planning task, s be a state and $s^{\mathrm{FDD}} = \mathrm{FDD}(s)$ the corresponding state in the compiled task. We have to show that $h^{max}(s) = h^{max}(s^{\mathrm{FDD}})$. The proof is analogous to the proof of Theorem 7. The only difference lies in the inductive case of the proof for

$$h^{max}_s(f) = \min_{a\in A(f)} \min_{p\in P(c_a)} (h^{max}_s(\mathrm{pre}(a) \cup p) + c_a(p)) = h^{max}_{s^{\mathrm{FDD}}}(f).$$

Let $a \in A(f)$ be an action in the original task achieving f. In the compiled task, instead of a single concluding action a^{eff} we have multiple concluding actions, each corresponding to a terminal node in the quasi-reduced, flattened EVMDD. Let $a^{\mathbf{w}} = \langle(\mathrm{aux}^a \doteq idx(\mathbf{w})), \mathrm{eff}(a) \wedge (\mathrm{aux}^a \doteq 0)\rangle$ be such a concluding action. Then, by Lemma 12 we have $h^{max}_{s^{\mathrm{FDD}}}(\mathrm{pre}(a^{\mathbf{w}})) = \min_{p\in P(\mathbf{w})} h^{max}_{s^{\mathrm{FDD}}}(s_p \cup \mathrm{pre})$. By construction of the flattened EVMDD, $c_{a^{\mathbf{w}}} = c_a(s_p)$, where s_p is a

partial state corresponding to a path from the root node to $\mathbf{w}$. Therefore, we get

$$h^{max}_{s^{\mathrm{FDD}}}(f) = \min_{a^{\mathbf{w}} \in A^{\mathrm{FDD}}(f)} \min_{p \in P(\mathbf{w})} \left(h^{max}_{s^{\mathrm{FDD}}}(s_p \cup \mathrm{pre}) \right) + c_a(s_p).$$

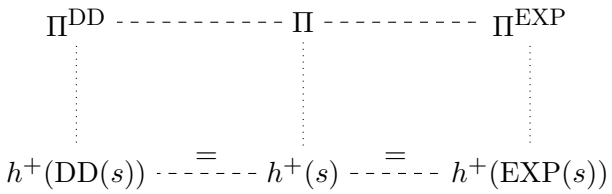
Since the two minimizations correspond, by construction, to minimizing over $p \in P(c_a)$ we can, with similar reasons to Theorem 7, apply the induction hypothesis and are done. □

This brings up an interesting question: does the additional information we get from the heuristic by representing the cost functions as flattened EVMDDs outweigh the overhead in representation size? If EVMDDs and flattened EVMDDs are of similar size, then it clearly makes sense to use flattened EVMDDs instead, at least when we use the maximum heuristic. But what if the size differs drastically? We will come back to this question when we empirically evaluate the maximum heuristic for both compilations (cf. Chapter 5).

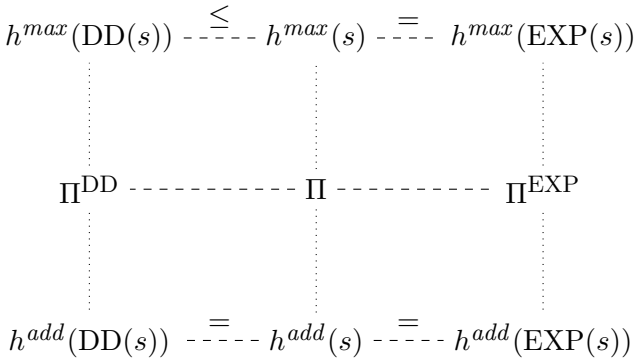
3.3 Summary

We conclude our theoretical evaluation of relaxation heuristics by summarizing the previous results in Figure 3.6. The optimal delete relaxation heuristic h^+ is invariant under both types, exponential and EVMDD compilation (Figure 3.6a). For the approximations, the additive heuristic is also invariant under both types of compilation, if the EVMDDs are quasi-reduced. The maximum heuristic, however, is only invariant under exponential compilation. For EVMDD compilation, we can only guarantee invariance if we rely on flattened quasi-reduced EVMDDs.

In the upcoming chapter, we introduce abstraction heuristics, which are usually more powerful than relaxation heuristics and also (sometimes) based on Cartesian sets.



(a) Invariance results for h^+.



(b) Invariance results for h^{add} and h^{max}.

Figure 3.6: Theoretical summary of Chapter 3.

CHAPTER 4

Abstraction Heuristics

While relaxation heuristics relax the problem by not considering negative effects of actions, abstraction heuristics rely on a transformation of the problem which only reflects some parts of the original problem by aggregating multiple states together into abstract states. There has been a plethora of research on abstractions for classical planning and different types of abstractions have been introduced over the years. *Pattern database* heuristics were initially introduced by Culberson and Schaeffer (1998) to compute admissible consistent heuristics for sliding tile puzzles and Rubik's Cube problems (Korf 1997) and later adopted by Edelkamp (2001) for general classical planning problems. They rely on so called *projection abstractions*, which can be seen as the "simplest" type of abstraction. *Cartesian abstractions* are more general than projection abstractions. Seipp and Helmert (2013) introduced Cartesian abstractions to planning by adapting the counterexample-guided abstraction refinement (CEGAR) algorithm (Clarke et al. 2000) to compute iteratively more fine-grained abstractions, resulting in potentially more accurate heuristics than what is possible with projection abstractions. Finally, *Merge-and-Shrink abstractions* (Dräger et al. 2009; Helmert et al. 2014) form an even more general class of abstractions than Cartesian abstractions, as every abstraction can be represented (not necessarily compactly) as a merge-and-shrink abstraction (Helmert et al. 2015). They rely on *factored transition systems* which are repeatedly *merged* by computing the product of two systems and *shrunk* by applying an abstraction to the product.

We start by giving the necessary definitions. In the following, we base our notation of abstractions on the work of Seipp and Helmert (2018) and extend it to transition systems with weighted transitions. As already noted, an abstraction aggregates multiple states into abstract states, but preserves transitions.

Definition 41 (Abstraction). Let Π be a planning task and $\mathcal{T}_\Pi = (\mathcal{S}, L, T, s_I, S_\star)$ the transition system induced by Π. An abstraction relation $\sim$ for Π is an

equivalence relation on $\mathcal{S}$. Equivalence classes of $\sim$ are called *abstract states*. Given a state s, we write $s^\sim$ for the equivalence class to which s belongs and call the function mapping s to $s^\sim$ the *abstraction function*.

The *abstract transition system* induced by $\sim$ is $\mathcal{T}_\Pi^\sim = (\mathcal{S}^\sim, L, T^\sim, {s_I}^\sim, S_\star^\sim)$, where

- $\mathcal{S}^\sim = \{s^\sim | s \in \mathcal{S}\}$,
- $T^\sim = \{s^\sim \xrightarrow{a, c_a(s)} s[a]^\sim \mid s \xrightarrow{a, c_a(s)} s[a] \in T\}$ and
- $S_\star^\sim = \{s^\sim | s \in S_\star\}$.

Sometimes $\mathcal{T}_\Pi^\sim$ is also called an *induced* abstraction, which can be understood as the abstraction does not introduce additional behaviour. In principle, we can also have a *non-induced* abstraction, where any of the components of $\mathcal{T}_\Pi^\sim$ (i.e. $\mathcal{S}^\sim$, L, $T^\sim$ or $S_\star^\sim$) is enlarged (Seipp and Helmert 2018), but we note that all of the abstractions we consider here are induced abstractions. Additionally, we note that an (induced) abstraction can have non-deterministic transitions, i.e. two transitions with the same label lead to different abstract successor states.

As we did already with relaxed states, we will call states of the original transition system *concrete* states and states in the abstract transition system *abstract* states. We will also say $s^\sim$ subsumes s', if ${s'}^\sim = s^\sim$, i.e. s and s' belong to the same abstract state. The definition of abstract transitions might seem straightforward but warrants discussion. For each transition between two states there is a transition between the corresponding abstract states with the weight corresponding to the weight of the original transition. In particular, this implies that multiple transitions between two abstract states may have different weights even if they are induced by the same action. Thus, in the worst case the abstract transition system has as many transitions as the concrete transition system. However, we will see that for some types of abstractions the transitions collapse even when actions have state-dependent action costs, and that for some other types of abstractions we can approximate the weights while still obtaining admissible estimates.

In the following, we will see an example of an abstraction applied on our logistics example task.

Example 18. Consider the logistics task in Example 1 and its transition system depicted in Figure 2.3. Figure 4.1 depicts an abstraction of the logistics task, such that states with matching domain values for the $\mathrm{p_1}$-at and $\mathrm{p_2}$-at variables belong to the same partition. Note that an optimal plan now simply consists of load and unload actions and has a cost of 4.

Based on abstract transition systems, we are now able to define abstraction heuristics.

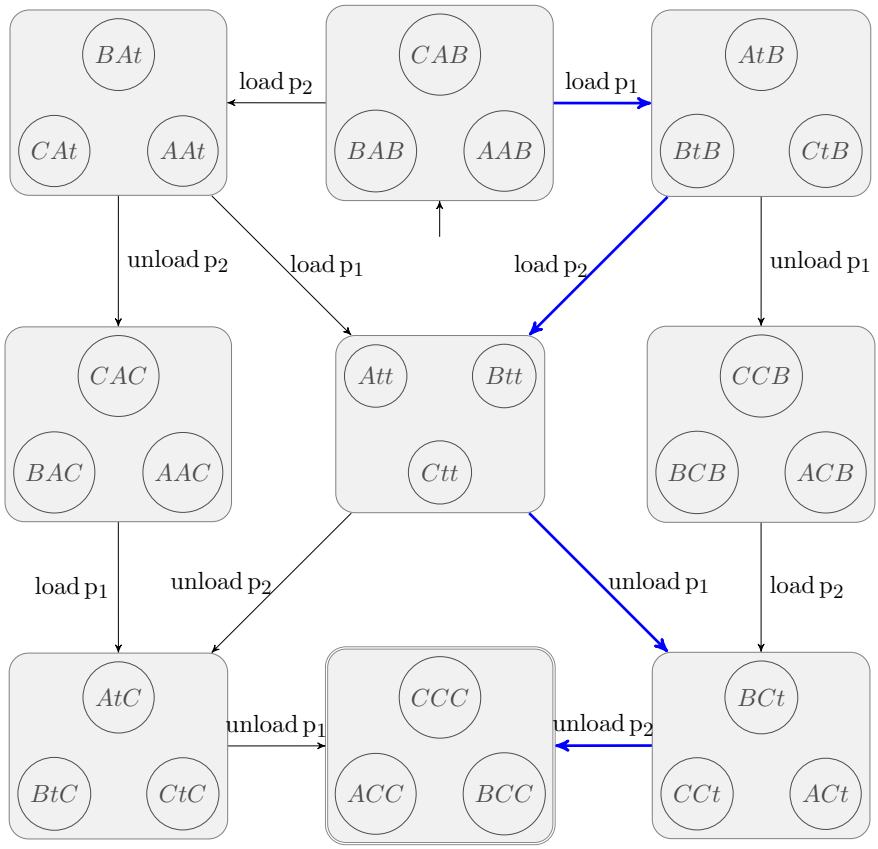


Figure 4.1: Transition system induced by an abstraction of the logistics task in Example 1. Round nodes depict concrete states, which are subsumed by abstract states, shown as rectangles. Self-loops are omitted. An optimal plan with cost 4 in the abstract transition system is highlighted in blue.

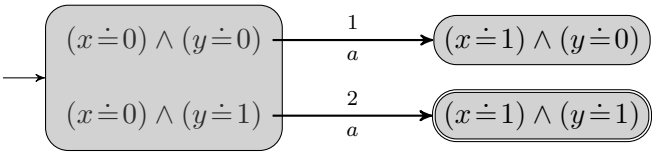
Definition 42 (Abstraction heuristic). Let Π be a planning task and $\sim$ be an abstraction relation. Then, the abstraction heuristic $h^{\sim}(s)$ is the cost of an optimal plan of $\mathcal{T}_{\Pi}^{\sim}$ starting in $s^{\sim}$, or ∞ if no such plan exists.

Abstraction heuristics are admissible heuristics. Before we discuss different types of abstractions, we give some results which hold for arbitrary abstractions. First, every abstraction heuristic is invariant under exponential compilation. The underlying reason is that the exponential compilation preserves transitions (cf. Lemma 4) and therefore each transition in the abstraction of the original task is also a transition in the abstraction of the compiled task and the other way around.

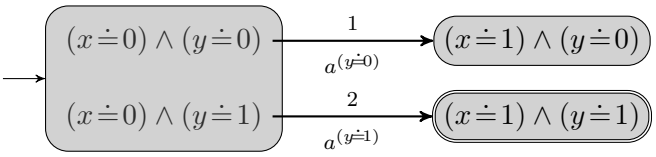
Example 19. Consider a planning task Π with two binary variables x, y and an action $a = \langle (x \doteq 0), (x \doteq 1) \rangle$ with $c_a = y + 1$. Let $s_I = (x \doteq 0) \wedge (y \doteq 1)$ and $s_\star = (x \doteq 1) \wedge (y \doteq 1)$. Now, consider an abstraction relation $\sim$, such that we

have the following abstract states: $s_1^\sim = \{(x \doteq 0) \wedge (y \doteq 0), (x \doteq 0) \wedge (y \doteq 1)\}$, $s_2^\sim = \{(x \doteq 1) \wedge (y \doteq 1)\}$, $s_3^\sim = \{(x \doteq 1) \wedge (y \doteq 0)\}$.

Then, we have $h^\sim(s_I) = 2 = h^\sim(\mathrm{EXP}(s_I))$. To see this, consider Figure 4.2. On the top, we have the abstract transition system induced by Π. On the bottom, we have the transition system induced by Π^{EXP}. Note that every transition from the original transition system is also contained in the transition system induced by the compilation, but the figure omits self-loops induced by drive actions. Additionally, note that we have non-deterministic transitions, i.e. two transitions with the same label which lead to different abstract successor states.



(a) Abstract transition system induced by Π.



(b) Abstract transition system induced by Π^{EXP}.

Figure 4.2: Abstract transition systems of Example 19. Abstract states are depicted as rectangles. Transitions are annotated with the corresponding action. Self-loop transitions are omitted.

Theorem 11. *Let Π be a planning task and $\sim$ an abstraction relation. Then, $h^\sim$ is invariant under exponential compilation.*

Proof. Follows from Lemma 4. Since both transition systems share the same structure, their abstract transition systems also share the same structure and therefore $h^\sim(s) = h^\sim(\mathrm{EXP}(s))$. □

Note that general abstraction heuristics are *not* invariant under EVMDD compilation. Depending on the type of abstraction they can yield arbitrarily worse heuristic estimates. To see this, consider that if the information of the auxiliary and semaphore variables are lost due to abstraction, only the cost of the concluding action applies. But even if the semaphore and auxiliary variables are preserved by the abstraction, invariance is only given if the abstraction obeys some restrictions. We will discuss this in more detail in the following sections, when we consider how projection and Cartesian abstractions behave under EVMDD compilation.

In principle, there are two requirements on an abstraction heuristic: first, the resulting heuristic should yield *informative estimates* and second, the heuristic should be *efficient to compute*. Strong abstraction heuristics fulfill both of these requirements. The question is, do classical planning abstractions still fulfill these requirements if we consider the more general setting of state-dependent action costs? With this question in mind, we will investigate two types of abstraction heuristics: projection abstractions and their generalization, Cartesian abstractions. Projection abstractions are the underlying type of abstractions used in pattern database heuristics. They not only serve as a good introduction to the topic of abstraction heuristics, but are also used in some of the strongest abstraction heuristics in the literature (Seipp and Helmert 2018). Cartesian abstractions generalize projection abstractions, and, as we will see, they are invariant under EVMDD compilation, if the underlying abstraction guarantees some properties. Therefore, the theoretical results obtained for Cartesian abstractions are immediately applicable to projection abstractions.

4.1 Projection Abstractions

Pattern database (PDB) heuristics were initially introduced by Culberson and Schaeffer (1998) to compute admissible consistent heuristics for sliding-tile puzzles and Rubik's Cube problems (Korf 1997) and later adopted by Edelkamp (2001) for general classical planning problems. Informally, a pattern database heuristic precomputes the goal distances of small abstract transition systems (based on the pattern) and stores them in a perfect hash table. The heuristic estimate of a concrete state is then retrieved by looking up the goal distance of the corresponding abstract state. Depending on the pattern, the combination of different patterns may yield an admissible heuristic estimate.

Patterns are based on so-called *projections*. A projection is an abstraction that fully preserves some variables (the pattern), and completely ignores the rest.

Definition 43 (Projection abstractions). Let Π be a planning task with variables $\mathcal{V}$ and let $\mathcal{P} \subseteq \mathcal{V}$. Let $s\!\upharpoonright_{\mathcal{P}}$ denote the partial variable assignment defined on $\mathcal{P}$ with $s\!\upharpoonright_{\mathcal{P}}(v) = s(v)$ for all $v \in \mathcal{P}$. The *projection* $\sim_{\mathcal{P}}$ is defined as $s_1 \sim_{\mathcal{P}} s_2$ if and only if $s_1\!\upharpoonright_{\mathcal{P}} = s_2\!\upharpoonright_{\mathcal{P}}$. We call $\mathcal{P}$ the *pattern* of $\sim_{\mathcal{P}}$. Furthermore, we write $h^{\mathcal{P}}$ for the projection abstraction heuristic $h^{\sim_{\mathcal{P}}}$.

We have already seen an example of a projection: the abstract transition system of Example 18 is induced by a projection abstraction, where $\mathcal{P} = \{\mathrm{p_1\text{-}at}, \mathrm{p_2\text{-}at}\}$. Then, the heuristic of the initial state $s_I = (\mathrm{t\text{-}at} \doteq A) \wedge (\mathrm{p_1\text{-}at} \doteq A) \wedge (\mathrm{p_2\text{-}at} \doteq B)$ is $h^{\mathcal{P}}(s_I) = 4$. A pattern database usually precomputes the goal distances of all abstract states for each pattern and stores them in a perfect hash table with look-up time linear in the abstract

state description length (Edelkamp 2001). With the above requirements on abstraction heuristics in mind, we can ask the two questions: what is a good selection of patterns (that yields informative estimates) and given a pattern, how can we efficiently compute the pattern database.

For classical planning tasks, the second question is answered by a particularly useful theorem for syntactic projections. The syntactic projection of a planning task Π regarding a pattern $\mathcal{P}$ is the planning task $\Pi\downarrow_{\mathcal{P}}$ where all references to state variables that are not contained in the pattern are removed from all parts of the task description. The theorem says that if Π is an SAS$^+$ task, then the transition system induced by $\Pi\downarrow_{\mathcal{P}}$ has the same structure as the abstract transition system $\mathcal{T}_{\Pi}^{\sim}$ induced by the projection $\sim_{\mathcal{P}}$.[1] The question is, whether we can reproduce this result for planning tasks with state-dependent action costs. For this, we have to formally define the syntactic projection of a planning task with state-dependent action costs.

Definition 44 (Syntactic Projection). Let $\Pi = (\mathcal{V}, \mathcal{A}, s_I, s_\star, c)$ be a planning task and let $\mathcal{P} \subseteq \mathcal{V}$. The *syntactic projection* $\Pi\downarrow_{\mathcal{P}}$ is the planning task $(\mathcal{P}, \mathcal{A}\downarrow_{\mathcal{P}}, s_I\downarrow_{\mathcal{P}}, s_\star\downarrow_{\mathcal{P}}, c\downarrow_{\mathcal{P}})$, where

- $\mathcal{A}\downarrow_{\mathcal{P}} = \{a\downarrow_{\mathcal{P}} | a \in \mathcal{A}\}$ with $a\downarrow_{\mathcal{P}} = \langle \mathrm{pre}\downarrow_{\mathcal{P}}, \mathrm{eff}\downarrow_{\mathcal{P}} \rangle$ for $a = \langle \mathrm{pre}, \mathrm{eff} \rangle$, and
- $c\downarrow_{\mathcal{P}}(a\downarrow_{\mathcal{P}}, s\downarrow_{\mathcal{P}}) = \min_{s' \in \mathcal{S} | s\downarrow_{\mathcal{P}} \subseteq s'} c(a, s')$ for all $s \in \mathcal{S}, a \in \mathcal{A}$.

The definition of the syntactic projection of an action is straightforward: we only consider variables included in the projection and ignore all other variables. Our definition of the syntactic projection of the action cost warrants explanation for two reasons: first, the projection of an action, $a\downarrow_{\mathcal{P}}$, is only defined as a tuple of partial variable assignments and has no identity. Therefore, two different concrete actions could formally result in a single abstract action, and we could not obtain the original action and in particular the original cost function required for the abstract cost function. A formally correct definition would therefore have to label the abstract actions, such that we can retrieve the original action from the labeling. We abuse notation instead, and just note that we can always retrieve the original action (and its cost function), since every projection of an action will result in a unique abstract action. Second, we define the cost of an action applied in an abstract state $s^\sim$ as the *minimum action cost* of all concrete states subsumed by $s^\sim$. This has an important implication: the transition system induced by $\Pi\downarrow_{\mathcal{P}}$ does not have the same structure as the abstract transition system $\mathcal{T}_{\Pi}^{\sim_{\mathcal{P}}}$, as the weights of the transitions differ (an example follows below). However, this definition has two useful properties: first, we can efficiently compute abstract costs by local minimization of the EVMDD corresponding to c_a, as abstract states in a projection

[1] This was stated by Sievers et al. (2012), but they do not give a reference. Proving this theorem is also a recurring exercise in the planning lecture of Malte Helmert (e.g. Helmert et al. 2017). The proof will be given in Theorem 12.

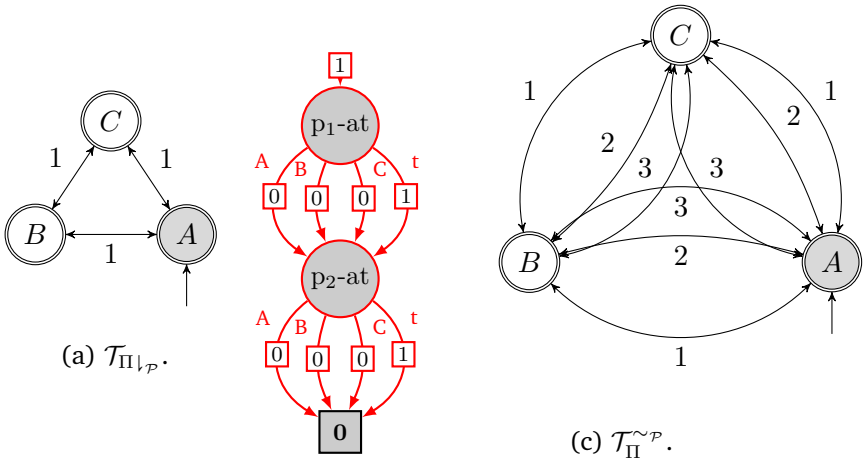
are Cartesian sets (cf. Theorem 3). And second, the minimum goal distances (i.e. optimal plans) of both (abstract) transition systems are still equal. As a result, if we are only interested in optimal plans, we can make use of the syntactic projection to compute a transition system with behaviour similar to the transition system induced by the projection.

Example 20. Consider the logistics planning task of Example 1 and the pattern $\mathcal{P} = \{\text{t-at}\}$, i.e. we only consider information about the truck. Then, we get the syntactic projection $\Pi\downarrow_{\mathcal{P}}$ with

- $\mathcal{P} = \{\text{t-at}\}$, $\mathcal{D}_{\text{t-at}} = \{A, B, C\}$
- $\mathcal{A} = \{\text{drive-AB} = \langle(\text{t-at} \doteq A), (\text{t-at} \doteq B)\rangle,$
  $\text{drive-BA} = \langle(\text{t-at} \doteq B), (\text{t-at} \doteq A)\rangle,$
  $\text{drive-AC} = \langle(\text{t-at} \doteq A), (\text{t-at} \doteq C)\rangle,$
  $\text{drive-CA} = \langle(\text{t-at} \doteq C), (\text{t-at} \doteq A)\rangle,$
  $\text{drive-BC} = \langle(\text{t-at} \doteq B), (\text{t-at} \doteq C)\rangle,$
  $\text{drive-CB} = \langle(\text{t-at} \doteq C), (\text{t-at} \doteq B)\rangle\},$
- $s_I = (\text{t-at} \doteq A)$,
- $s_\star = \top$,
- $c_a(s^\sim) = \min_{s \in s^\sim} ([\text{p}_1\text{-at}(s) = t] + [\text{p}_2\text{-at}(s) = t] + 1)$ for all actions $a \in \mathcal{A}$.

Note that we omit actions without effect (i.e. load and unload actions) from the task description. To compute the action cost function we can make use of the EVMDD of the original action cost function. Whenever we want to compute the cost of an abstract state $s^\sim$, we locally minimize the corresponding EVMDD. Figure 4.3a depicts the transition system induced by $\Pi\downarrow_{\mathcal{P}}$. Figure 4.3b depicts an example of the local minimization of the associated cost EVMDD. Since all abstract states include a state where both packages are not in the truck the cost of each drive action for each abstract state is 1. Finally, Figure 4.3c depicts the abstract transition system of the original task. Note that we have multiple outgoing transitions, but a transition with minimum cost always has a corresponding transition in $\mathcal{T}_{\Pi\downarrow_{\mathcal{P}}}$. Self-loops are omitted.

Obviously, we don't have the same result as we have for classical planning, since transitions in the abstract transition system of the original task have different weights than transitions in the transition system induced by the syntactic projection. Apart from the weights, however, both transition systems have the same structure. Moreover, the definition of the abstract cost function implies that both transition systems admit the same optimal plans.



(a) $\mathcal{T}_{\Pi\downarrow_\mathcal{P}}$.

(b) EVMDD of c_{drive}.

(c) $\mathcal{T}_\Pi^{\sim_\mathcal{P}}$.

Figure 4.3: Depiction of Example 20

Theorem 12 (Syntactic projection vs. projection). *Let Π be a planning task and let $\mathcal{P}$ be a pattern for Π. Then, $\mathcal{T}_{\Pi\downarrow_\mathcal{P}}$ has the same structure as $\mathcal{T}_\Pi^{\sim_\mathcal{P}}$ apart from weights. Moreover, each optimal plan of $\Pi\downarrow_\mathcal{P}$ is an optimal plan in $\mathcal{T}_\Pi^{\sim_\mathcal{P}}$, and the other way around.*

Proof. Recall that $\Pi\downarrow_\mathcal{P} = (\mathcal{P}, \mathcal{A}\downarrow_\mathcal{P}, {s_I}\downarrow_\mathcal{P}, {s_\star}\downarrow_\mathcal{P}, c\downarrow_\mathcal{P})$ and $\mathcal{T}_\Pi^\sim = (\mathcal{S}^\sim, L, T^\sim, {s_I}^\sim, S_\star^\sim)$, where $\sim$ denotes $\sim_\mathcal{P}$. We denote members of $\Pi\downarrow_\mathcal{P}$ with a prime, e.g. $\mathcal{S}'$, L' and T' denote states, labels and transitions in the transition system induced by $\Pi\downarrow_\mathcal{P}$. We now show that there is a bijection $\varphi : \mathcal{S}' \to \mathcal{S}^\sim$ with

(a) $\varphi({s_I}\downarrow_\mathcal{P}) = {s_I}^\sim$,

(b) ${s_\star}\downarrow_\mathcal{P} \subseteq s' \in \mathcal{S}'$ if and only if $\varphi(s') \in S_\star^\sim$,

(c) There is a transition $s' \xrightarrow{a',w} t' \in T'$ for some $a' \in L'$ and some weight w if there is a transition $\varphi(s') \xrightarrow{a,w} \varphi(t') \in T^\sim$ for some $a \in L$,

(d) There is a transition $\varphi(s') \xrightarrow{a,w} \varphi(t') \in T^\sim$ for some $a \in L$ and some weight w if there is a transition $s' \xrightarrow{a',w'} t' \in T'$ for some $a' \in L'$ and some weight $w' \leq w$.

For φ, we choose the identity function $\varphi(s') = s'$:

(a) By construction, since $s^\sim = s\downarrow_\mathcal{P}$ we have ${s_I}^\sim = {s_I}\downarrow_\mathcal{P}$.

(b) For sufficiency, let there be a state $s' \in \mathcal{S}'$ such that ${s_\star}\downarrow_\mathcal{P} \subseteq s'$. We can construct a concrete state s such that $s' = s\downarrow_\mathcal{P}$ and $s_\star \subseteq s$. We choose

$s(v) = s_\star(v)$ if $v \in \mathcal{P}$, $s(v) = d$ if $v \notin \mathcal{P}$ and $(v \doteq d) \in s_\star$, and arbitrary state values otherwise. Clearly, $s_\star \subseteq s$ and therefore $s\restriction_{\mathcal{P}} \in S_\star^\sim$.

For necessity, let $s' \in S_\star^\sim$. Then, there is a concrete state $s \in S_\star$ with $s\restriction_{\mathcal{P}} = s'$ and $s_\star \subseteq s$ and thus also $s_\star\restriction_{\mathcal{P}} \subseteq s$. Since s agrees with $s\restriction_{\mathcal{P}}$ on variables in $\mathcal{P}$ we have $s_\star\restriction_{\mathcal{P}} \subseteq s\restriction_{\mathcal{P}}$ and thus $s_\star\restriction_{\mathcal{P}} \subseteq s'$.

(c) Let $s' \xrightarrow{a',w'} t' \in T'$. Then, we have an action $a \in \mathcal{A}$ and states $s, t \in \mathcal{S}$ such that $s\restriction_{\mathcal{P}}[a\restriction_{\mathcal{P}}] = t\restriction_{\mathcal{P}}$ with $s\restriction_{\mathcal{P}} = s'$, $t\restriction_{\mathcal{P}} = t'$ and $c(a,s) = \min_{s' \in \mathcal{S} | s\restriction_{\mathcal{P}} \subseteq s'} c(a,s') = w'$. Similar to before, we now construct a concrete state s'' such that

$$s''(v) = \begin{cases} s(v) & \text{if } v \in \mathcal{P} \cup \mathrm{vars}(c_a) \\ d & \text{if } v \notin \mathcal{P} \text{ and } (v \doteq d) \in \mathrm{pre}(a) \\ \text{arbitrary} & \text{otherwise.} \end{cases}$$

s'' is well-defined, as we require $\mathrm{vars}(\mathrm{pre}(a))$ to be disjoint with $\mathrm{vars}(c_a)$ (Assumption 3). Clearly, a is applicable in s'' since $a\restriction_{\mathcal{P}}$ is applicable in $s\restriction_{\mathcal{P}}$ and also $c(a,s'') = w'$, since s'' agrees with s on the variables in the cost function of a. Now let $t'' = s''[a]$. For variables $v \in \mathcal{P}$ which are not affected by a we have $t''(v) = s''(v) = s(v) = s\restriction_{\mathcal{P}} = t\restriction_{\mathcal{P}} = t$ since $a\restriction_{\mathcal{P}}$ does also not affect v. For variables $v \in \mathcal{P}$ where a has an effect with $(v \doteq d)$ we have $t''(v) = d = t\restriction_{\mathcal{P}}(v) = t(v)$, since $a\restriction_{\mathcal{P}}$ also contains the effect $(v \doteq d)$. Therefore $s'' \xrightarrow{a,w'} t'' \in T$ and thus $s''\restriction_{\mathcal{P}} \xrightarrow{a,w'} t''\restriction_{\mathcal{P}} \in T^\sim$. Since $s''\restriction_{\mathcal{P}} = s\restriction_{\mathcal{P}} = s'$ and $t''\restriction_{\mathcal{P}} = t\restriction_{\mathcal{P}} = t'$ we have $s' \xrightarrow{a,w'} t' \in T^\sim$.

(d) Let $s' \xrightarrow{a,w} t' \in T^\sim$. Then, there exists an action $a = \langle \mathrm{pre}, \mathrm{eff} \rangle \in \mathcal{A}$ and states $s, t \in \mathcal{S}$ such that $s[a] = t$ with $s\restriction_{\mathcal{P}} = s^\sim = s'$, $t\restriction_{\mathcal{P}} = t^\sim = t'$ and $w = c(a,s)$. Now, consider action $a' = a\restriction_{\mathcal{P}} = \langle \mathrm{pre}\restriction_{\mathcal{P}}, \mathrm{eff}\restriction_{\mathcal{P}} \rangle \in \mathcal{A}\restriction_{\mathcal{P}}$. Since a is applicable in s we have $\mathrm{pre} \subseteq s$ and also $\mathrm{pre}\restriction_{\mathcal{P}} \subseteq s$. Furthermore, $\mathrm{pre}\restriction_{\mathcal{P}}$ only contains variables in $\mathcal{P}$, therefore $\mathrm{pre}\restriction_{\mathcal{P}} \subseteq s\restriction_{\mathcal{P}}$. Since we assume that there are no trivially inapplicable actions, $a\restriction_{\mathcal{P}}$ is therefore also applicable in s'. Furthermore, for $v \in \mathcal{P}$, we have

$$\begin{aligned} &a' \text{ has no effect } (v \doteq d) \Leftrightarrow a \text{ has no effect } (v \doteq d) \Leftrightarrow t(v) = s(v) \text{ and} \\ &a' \text{ has an effect } (v \doteq d) \Leftrightarrow a \text{ has an effect } (v \doteq d) \Leftrightarrow t(v) = d. \end{aligned}$$

Together with $t' = t\restriction_{\mathcal{P}}$ we thus get $t(v) = t'(v)$ and therefore $s' \xrightarrow{a\restriction_{\mathcal{P}},w'} t' \in T'$. Furthermore, we have $w' = c(a\restriction_{\mathcal{P}}, s\restriction_{\mathcal{P}}) = \min_{s' \in \mathcal{S} | s\restriction_{\mathcal{P}} \subseteq s'} c(a, s')$ and thus $w' \leq w = c(a,s)$.

Therefore, a transition in $\Pi\downarrow_{\mathcal{P}}$ with weight w always has a corresponding transition in $\mathcal{T}_{\Pi}^{\sim_{\mathcal{P}}}$ with weight w. Moreover, there is no cheaper transition $\mathcal{T}_{\Pi}^{\sim_{\mathcal{P}}}$ between the same states, therefore $\Pi\downarrow_{\mathcal{P}}$ and $\mathcal{T}_{\Pi}^{\sim_{\mathcal{P}}}$ admit the same optimal plans. □

Given a pattern $\mathcal{P}$, we now have an efficient way (given that the cost EVMDDs are compact) to generate the abstract transition system induced by $\sim_{\mathcal{P}}$. The question is how this result enables us to compute the pattern database for a given pattern $\mathcal{P}$. As we already mentioned, a pattern database stores the shortest goal distances of all abstract states in a hash table. For this, efficient implementations of PDB heuristics perform a search of the abstract state space in backward direction, starting from the goal states of the task (Sievers et al. 2012). This can be efficiently done with Dijkstra's algorithm (Dijkstra 1959), which is a special case of the $A^\star$ algorithm, where $h(s) = 0$ for all states s. Sievers et al. (2012) describe two implementations of PDB heuristics for classical planning tasks.

The baseline implementation first constructs the abstract state space (with backward edges) by relying on the syntactic projection and uses Dijkstra's algorithm to compute abstract goal distances afterwards. This can be done quite efficiently, by starting from the goal states of the task and searching towards all other states. These distances are then stored in a hash table with a perfect hash function, called the *rank* of an abstract state. Our result for the syntactic projection of tasks with state-dependent action costs thus enables us to implement PDB heuristics in the same way.

However, the second implementation described by Sievers et al. (2012) vastly outperforms the baseline implementation for PDBs with a large number of abstract states. Their approach can be summarized in three steps: first, they perform a precompilation step to make all actions injective, i.e. to enforce that whenever an action has variable v in its effect, then v also is contained in the precondition. This allows them to compute the *regression*[2] (backward application) of an action a via forward application of an action $\hat{a}$ which corresponds to the backward application of a. The second step uses a data structure called *successor generator* (Helmert 2006b, p. 216) to efficiently compute the predecessor states of any state expanded during Dijkstra's algorithm. Finally, they avoid generation of states altogether, by immediately computing the hash value (the rank) of a successor state, given the hash value of the predecessor state and action $\hat{a}$. We could now discuss how to adapt their approach to tasks with state-dependent action costs. This would require us to define the action cost function of the "backward" action $\hat{a}$ and also to come up with a way to compute the action cost given just the hash value of a state, instead of the state itself. But do we really have to do this? What if we can instead rely once again on the compilation of state-dependent action costs and just apply the

[2]The regression of action a and state s' computes the set of predecessor states which lead to s' by application of a.

PDB heuristic computation on the compiled task? To answer this question, we have to consider if the PDB heuristic is invariant under exponential and EVMDD compilation.

From Theorem 11 we already know that every abstraction heuristic is invariant under exponential compilation. The obvious question is now if we can achieve a similar result for EVMDD compilation. While the exponential compilation does not modify the variables of the original task, EVMDD compilation adds auxiliary and semaphore variables, which are necessary to guarantee sequential application of intermediate actions. It comes to no surprise that without these variables we lose important information.

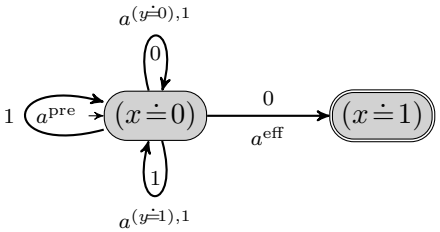
Example 21. Consider again the planning task Π of Example 19 with two binary variables x, y and an action $a = \langle (x \doteq 0), (x \doteq 1) \rangle$ with $c_a = y + 1$, $s_I = (x \doteq 0) \wedge (y \doteq 1)$ and $s_\star = (x \doteq 1) \wedge (y \doteq 1)$. In the EVMDD compilation, we get the following actions:

$$
\begin{aligned}
a^{\text{pre}} &= \langle (\text{aux} \doteq 0) \wedge (\text{lock} \doteq 0) \wedge (x \doteq 0), (\text{aux} \doteq 1) \wedge (\text{lock} \doteq 1) \rangle, \mathit{cost} : 1 \\
a^{(y \doteq 0),1} &= \langle (\text{aux} \doteq 1) \wedge (y \doteq 0), (\text{aux} \doteq 2) \rangle, \mathit{cost} : 0 \\
a^{(y \doteq 1),1} &= \langle (\text{aux} \doteq 1) \wedge (y \doteq 1), (\text{aux} \doteq 2) \rangle, \mathit{cost} : 1 \\
a^{\text{eff}} &= \langle (\text{aux} \doteq 2), (x \doteq 1) \wedge (\text{aux} \doteq 0) \wedge (\text{lock} \doteq 0) \rangle, \mathit{cost} : 0
\end{aligned}
$$

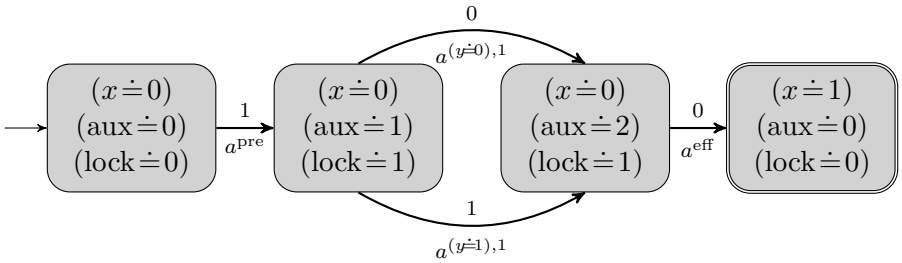
Now, consider the pattern $\mathcal{P} = \{x\}$. Figure 4.4a depicts the abstract transition system induced by $\sim_{\mathcal{P}}$. Since auxiliary facts are not preserved by the abstraction, we have an action which immediately achieves $(x \doteq 1)$ with cost 0.

Obviously, this example shows that in general, PDB heuristics are not invariant under EVMDD compilation. But, what if we can achieve some form of quasi-invariance by lifting the pattern to the auxiliary and semaphore variables? Figure 4.4b depicts the abstract transition system of the previous example induced by the pattern $\mathcal{P}_{\text{DD}} = \{x, \text{aux}, \text{lock}\}$. The cost of reaching a goal state is 1, which corresponds to $h^{\sim}(s_I)$. As it turns out, this result is generally applicable for PDB heuristics. The proof relies on Theorem 15 about Cartesian abstractions, which we will introduce later when we generalize projection heuristics. Strictly speaking, we could have avoided this forward reference if we introduced Cartesian abstractions first, but since projection abstractions are easier to convey they serve as a better starting point to abstractions in general. As Section 4.2 does not rely on the definitions and results given here the ambitious reader may also skip to Section 4.2 and come back after the proof of Theorem 15.

Theorem 13 (PDB heuristics are quasi-invariant under EVMDD compilation). *Let Π be a planning task and let $\mathcal{P}$ be a pattern for Π. Let $\mathcal{P}_{\text{DD}} = \mathcal{P} \cup \{\text{lock}\} \cup \{\text{aux}^a | a \in \mathcal{A}\}$. Then, $h^{\mathcal{P}_{\text{DD}}}(\text{DD}(s)) = h^{\mathcal{P}}(s)$ for all $s \in \mathcal{S}$ of Π.*



(a) Abstract transition system induced by $\sim_{\mathcal{P}}$.



(b) Abstract transition system induced by $\sim_{\mathcal{P}_{\text{DD}}}$.

Figure 4.4

Proof. By definition, projection heuristics are Cartesian. Furthermore, from Theorem 12 we know that the transition system of the syntactic projection $\mathcal{T}_{\Pi\downarrow_{\mathcal{P}}}$ has the same structure as $\mathcal{T}_{\Pi}^{\sim}$. In particular, this implies that there are no transitions corresponding to non-deterministic actions (since $\mathcal{T}_{\Pi\downarrow_{\mathcal{P}}}$ is a planning task and planning tasks have no non-deterministic actions). Then, it follows from Theorem 15 that $h^{\mathcal{P}_{\text{DD}}}(\text{DD}(s)) = h^{\mathcal{P}}(s)$ for all $s \in \mathcal{S}$ of Π. □

This is an important result, as it allows us to apply the efficient implementation of Sievers et al. (2012) on the EVMDD compilation by use of $\mathcal{P}_{\text{DD}}$. There is one caveat though: EVMDD compilation results in $|\mathcal{A}|+1$ additional variables. Since the abstract transition system grows with the number of variables included in the pattern, $\mathcal{P}_{\text{DD}}$ would result in an exponential blow-up of the abstract transition system. Fortunately, we can circumvent this problem by applying the variable-compact EVMDD compilation (cf. Definition 28) instead. What is left to discuss is our first question: what is a good collection of patterns that yields informative heuristic estimates?

In their work about iterative construction of pattern databases, Haslum et al. (2007) establish some results about the heuristic functions produced by different collections of patterns. Recall (Def. 10) that a heuristic h dominates a heuristic h', if and only if $h(s) \geq h'(s)$ for all $s \in \mathcal{S}$.

Proposition 1 (Edelkamp 2001; Haslum et al. 2007). *Let $A \subseteq \mathcal{V}$ and $B \subseteq \mathcal{V}$ be two patterns. The heuristic $h(s) = \max(h^A(s), h^B(s))$ is admissible and dominates both h^A and h^B. Furthermore, if $A \subseteq B$ then h^B dominates h^A.*

If the set of actions that affect some variable in A is disjoint from the set of actions that affect any variable in B, then the heuristic $h(s) = h^A(s) + h^B(s)$ is also admissible. In this case we say that the patterns are additive[3]. *A set of patterns is additive if and only if all patterns in the set are pairwise additive.*

The additivity property is an important criterion, as it allows to combine the heuristic estimates of multiple projection abstraction heuristics to form a more powerful admissible heuristic and was already important in the initial work of Edelkamp (2001) on PDBs for classical planning. Based on the above properties, Haslum et al. (2007) propose the *canonical heuristic function* h^C of a pattern collection (i.e. a set of patterns) C.

Definition 45 (Canonical heuristic). Let s be a state, C a set of patterns, and let A be the set of all maximal (with regard to set inclusion) additive subsets of C. The *canonical heuristic function* of C is

$$h^C(s) = \max_{S \in \mathsf{A}} \sum_{\mathcal{P} \in S} h^{\mathcal{P}}(s).$$

The canonical heuristic is an admissible heuristic for classical planning tasks. Therefore, it is also admissible for the EVMDD compilation. However, there is one subtle issue: given two additive patterns A and B, the above condition of additivity does not consider the patterns A^{DD} and B^{DD} to be additive anymore. The reason for this is that each initial action a^{pre} affects the semaphore variable lock (or alternatively, the auxiliary variable aux), which is contained in both patterns A^{DD} and B^{DD}. Thus, when lifting the patterns to the additional variables, we can't use the additivity criterion to detect their additive behaviour. As a result, a direct search for sets of patterns on the compiled task will not achieve strong canonical heuristic estimates, as most of the patterns won't be regarded as additive. Fortunately, the result of Theorem 13 implies that the canonical heuristic can be lifted to the EVMDD compilation:

Corollary 6. *Let s be a state, C a set of patterns, and let* A *be the set of all maximal (with regard to set inclusion) additive subsets of C. Then,*

$$h^C(s) = \max_{S \in \mathsf{A}} \sum_{\mathcal{P} \in S} h^{\mathcal{P}_{\mathrm{DD}}}(\mathrm{DD}(s)).$$

Proof. Follows from Theorem 13, since $h^{\mathcal{P}_{\mathrm{DD}}}(\mathrm{DD}(s)) = h^{\mathcal{P}}(s)$. □

Thus, a set of patterns which results in a strong canonical heuristic estimate can be lifted to the auxiliary and semaphore variables and results in a strong heuristic estimate for the compiled task. What is left is to discuss how to compute such a set of patterns. We hope to be able to apply the techniques by Haslum et al. (2007) on tasks with state-dependent action costs,

[3] It is important to note that this condition is only sufficient for additivity, not necessary.

generate a collection of patterns for the original task, and lift this pattern to compute PDB heuristics on the compiled task. We briefly review the approach of Haslum et al. (2007) to compute sets of patterns and also describe the difference to the computation performed by Fast Downward (Helmert 2006b), the planner used in later experiments.

Their approach to compute sets of patterns is an iterative hill-climbing search in the search space of collections of patterns, i.e. each state in the search space corresponds to a pattern collection and the neighbours of a state are defined by modifications to the collection represented by the current state. More precisely, a neighbour of a state representing a collection of patterns $C = \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ is constructed by selecting a pattern $\mathcal{P}_i \in C$, a variable $v \notin \mathcal{P}_i$, and adding the new pattern $\mathcal{P}_{k+1} = \mathcal{P}_i \cup \{v\}$.

The initial state of the search is the collection with one pattern for each goal variable and in each iteration the neighbourhood of expanded pattern collections is evaluated and the best neighbour is chosen to be the current collection in the next iteration. The search ends if either the size limit is reached (which is a parameter of the hill climbing search) or if the current collection does not result in significant improvement. Evaluating the neighbourhood, i.e. ranking the relative quality of the pattern collections plays an important part in this search. Haslum et al. (2007) base their method to evaluate the neighbourhood on a formula to estimate the number of nodes expanded by tree search developed by Korf et al. (2001). Given a collection C, they evaluate the neighbour C' of C by sampling a number of states (from the planning problem) and count for how many states C' provides a better heuristic estimate than C. To sample states, they use random walks up to depth d, where d is estimated by the current heuristic and adjusted by the average cost/depth ratio. Additionally, they avoid redundant evaluation by performing a static and statistical analysis on which variables to add to improve the heuristic value of a pattern $\mathcal{P}$. The implementation in Fast Downward differs in some minor details (Fast Downward website 2018). For our case, the most important difference is how the depth d is estimated. In this implementation, the estimation is calculated by dividing the current heuristic estimate for the initial state (multiplied by some factor c to account for heuristic inaccuracy) by the average action costs of the planning task. For example, if $h(s_I) = 10$ and the average action cost is 2, then $d = c \cdot 10/2 = c \cdot 5$, i.e. the search samples states up to depth $5c$.

The summary of this approach reveals two important parts we have to consider when we want to apply this technique to EVMDD compiled tasks. First, the initial state of the pattern collection search consists of one pattern for each goal variable. In the compilation, this would mean that we have a pattern consisting of lock and a pattern consisting of aux. However, by Theorem 13 we already know that given a pattern $\mathcal{P}$ with $\mathrm{aux} \notin \mathcal{P}$ and $\mathrm{lock} \notin \mathcal{P}$ the PDB heuristic is only invariant if we consider $\mathcal{P} \cup \{\mathrm{lock}, \mathrm{aux}\}$. Therefore, for EVMDD compilation we will initialize the pattern search with the patterns

$\{v, \mathrm{lock}, \mathrm{aux}\}$ for each goal variable v and adapt the additivity check such that lock and aux are always considered additive with the original variables. What is left is to discuss how the depth of the random walk is estimated. First, note that our theorem about heuristic invariance only holds for states in the EVMDD compilation which have a corresponding state in the original task (i.e. states where auxiliary and semaphore values are 0). However, when sampling states in the compiled problem, we might end up at an intermediate state. In principle, we do not care about heuristic estimates of intermediate states (as search itself works on the original task, and only the heuristic is applied to the compiled task). Therefore, whenever we sample an intermediate state, we apply subsequent actions until the state corresponds to an original state. Furthermore, taking the average action costs in the EVMDD compilation may have different effects on the depth, depending on the original cost function. Consider the previous Example 21. In the EVMDD compilation, we have 4 actions resulting in an average action cost of 0.5. For simplicity, let us assume the heuristic is perfect, i.e. the goal distance of the initial abstract state $(x \doteq 0) \wedge (\mathrm{aux} \doteq 0) \wedge (\mathrm{lock} \doteq 0)$, which is 1, is the optimal plan cost, and we do not account for heuristic inaccuracy (i.e. $c = 1$). Then, the depth where a state is sampled will be $d = 1/0.5 = 2$. However, if we increase the number of domain values for y, then the average action cost will rise and with it depth d, even though the action costs are not relevant in this case.

In general, it is not easy to say what a "good" depth setting is. Even in the constant cost case, dividing by the average action cost might underestimate the depth by a fair amount if an optimal plan mostly consists of the application of cheap actions. Furthermore, even for an accurate depth estimate, the hill-climbing search might get stuck in local optima, and thus terminating the search for additional patterns early. Scherrer et al. (2015) indeed show that by extending the search beyond the current horizon (thus yielding larger pattern collections), performance of the heuristic can be improved. Such an approach might also be of use when the average depth based on the EVMDD compilation results early in a local optima. With this we conclude our discussion of PDB heuristics, and note that the empirical evaluation given in Chapter 5 includes a comparison of different hill-climbing configurations.

4.2 General Cartesian Abstractions

Now that we have achieved some promising results for projection abstractions we want to investigate a more general class of abstractions: Cartesian abstractions. The term Cartesian abstraction was coined in the model-checking literature by Ball et al. (2001) and, as we already mentioned, Cartesian abstractions generalize projection abstractions. We will first discuss how general Cartesian abstractions behave under EVMDD compilation and then once again investigate how to efficiently generate Cartesian abstractions which result in

strong heuristic estimates.

Definition 46 (Cartesian abstractions). An abstraction induced by $\sim$ is called Cartesian if all its abstract states are Cartesian sets and we then say $\sim$ is a *Cartesian abstraction relation*. We call domain values $d \in D_i$ consistent with an abstract state $s^\sim$, and also refer to D_i as $s^\sim(v_i)$.

We have already seen that relaxed states are also Cartesian sets. The important difference to Cartesian abstractions is that delete relaxation does not partition the state space, i.e. the delete relaxed state space is not induced by an equivalence relation $\sim$. Note that the abstraction given in Example 19 is Cartesian. Moreover, if we apply the EVMDD compilation on this example, we will see that arbitrary Cartesian abstractions are not invariant under EVMDD compilation, even if we preserve auxiliary and semaphore variables.

Theorem 14 (Cartesian abstraction heuristics are not invariant under EVMDD compilation). *Let Π be a planning task and $\sim$ a Cartesian abstraction relation. Then, $h^\sim$ is not invariant under EVMDD compilation even if $\sim$ does not aggregate the semaphore and auxiliary variables in the compiled task.*

Proof. Consider Example 19 and the associated EVMDD compilation, which is shown in Example 18. Figure 4.5 depicts the abstract transition system of the compiled task. As we can see, $h^\sim(\mathrm{DD}(s_I)) = 1$, while $h^\sim(s_I) = 2$. □

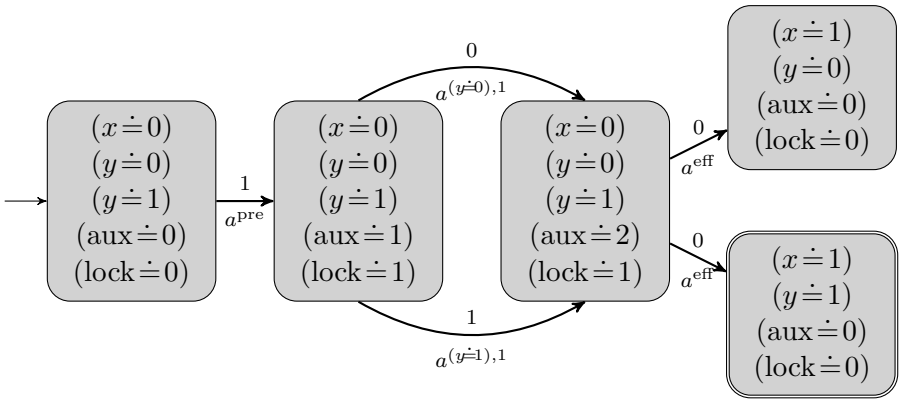


Figure 4.5: Abstract transition system of the EVMDD compilation of Example 19.

In principle, the reason why the heuristic is not invariant under EVMDD compilation is that the abstraction induces non-deterministic actions. This non-determinism is only reflected in the concluding action a^{eff}, but not in the intermediate actions which are responsible for the different action cost in the original transition system. Therefore, path costs in the abstract transition system of the compiled task may be cheaper than in the original abstract

transition system. Indeed, if all non-deterministic transitions in the induced abstract transition system of the original task do not differ in their weight, then the abstraction heuristic is invariant under EVMDD compilation.

Theorem 15. *Let Π be a planning task and $\sim$ a Cartesian abstraction. Let $\sim_{\mathrm{DD}}$ be the corresponding abstraction in Π^{DD} which does not aggregate semaphore and auxiliary variables. If $\mathcal{T}_{\Pi}^{\sim}$ does not contain any two transitions $s \xrightarrow{a,w} t$ and $s \xrightarrow{a,w'} t'$ with $w \neq w'$, then $h^{\sim_{\mathrm{DD}}}(\mathrm{DD}(s)) = h^{\sim}(s)$ for all $s \in \mathcal{S}$ of Π.*

Proof. A similar statement was shown recently by Bergdoll (2018) in his Master's thesis, who investigates a generalization of the EVMDD compilation that additionally transforms planning tasks with conditional effects to classical planning tasks. He shows that $h^{\sim_{\mathrm{DD}}}$ produces the same estimates as $h^{\sim}$ on the exponential compilation, if $\mathcal{T}_{\Pi}^{\sim}$ does not contain any non-deterministic transitions. While our assumptions are less strict (we only require equal weights for non-deterministic transitions), the generalization requires only some small adaptations.

From Lemma 5 we know that for every transition in Π leading from s to t with weight w there is a sequence of transitions in Π^{DD} leading from $\mathrm{DD}(s)$ to $\mathrm{DD}(t)$ with labels $\langle a^{\mathrm{pre}}, \ldots, a^{\mathrm{eff}} \rangle$ and sum of weights equal to w, and the other way around. We now show that this also holds in the abstract transition system, i.e. there is a transition $s^{\sim} \xrightarrow{a,w} t^{\sim}$ if and only if there is a sequence of transitions from $\mathrm{DD}(s)^{\sim}$ to $\mathrm{DD}(t)^{\sim}$ with sum of weights w.

Sufficiency is straightforward. Since every abstract transition in $\mathcal{T}_{\Pi}^{\sim}$ is induced by a transition in $\mathcal{T}_{\Pi}$ with label a we also have a sequence of transitions induced by action a in $\mathcal{T}_{\Pi^{\mathrm{DD}}}$ with sum of weights w (Lemma 5) and since semaphore and auxiliary variables are preserved by $\sim_{\mathrm{DD}}$ this sequence also exists in $\mathcal{T}_{\Pi^{\mathrm{DD}}}^{\sim_{\mathrm{DD}}}$.

For necessity, we have to show that if there is a sequence of transitions in $\mathcal{T}_{\Pi^{\mathrm{DD}}}^{\sim_{\mathrm{DD}}}$ with labels $\langle a^{\mathrm{pre}}, \ldots, a^{\mathrm{eff}} \rangle$ and sum of weights w between two non-intermediate states $\hat{s}$ and $\hat{t}$ (i.e. semaphore and auxiliary values are 0), then there exists a corresponding transition in $\mathcal{T}_{\Pi}^{\sim}$ with label a and weight w.

Let s be a state in $\mathcal{T}_{\Pi}$, such that $\mathrm{DD}(s) \in \hat{s}$ and a^{pre} is applicable in $\mathrm{DD}(s)$. Then, the original action a which induces a^{pre} is, by construction of the compilation, applicable in s and leads to state t with weight $w' = c_a(s)$. Since $\sim$ behaves exactly like $\sim_{\mathrm{DD}}$ for variables not introduced by the compilation we have that $t \in t^{\sim}$ if and only if $\mathrm{DD}(t) \in \hat{t}$. With this, we have established that there is a corresponding transition in $\mathcal{T}_{\Pi}^{\sim}$. What is left is to show that the weight of this transition is w, i.e. $w = w' = c_a(s)$. We show this by contraposition.

Assume this is not the case. That means there exists another sequence of actions $\pi = \langle a^{\mathrm{pre}}, \ldots, a^{\mathrm{eff}} \rangle$, starting from $\hat{s}$ leading to $\hat{t}$, and sum of weights equal to w'. This is possible for two reasons: first, due to non-deterministic

transitions the intermediate actions may correspond to the application of a which leads to a state subsumed by another abstract state, i.e. there exists a state $s' \in \hat{s}$ such that $s'[a] \in \hat{t}' \neq \hat{t}$. However, if $w \neq w'$ then this would mean that $\mathcal{T}_\Pi^\sim$ contains two transitions starting from the same abstract state with the same action label and different weights, which contradicts the initial assumption.

The second reason such a sequence may exist that without loss of generality there are two intermediate actions $a_1 = \langle(\text{aux} \doteq i) \wedge (x \doteq p), (\text{aux} \doteq j)\rangle$ and $a_2 = \langle(\text{aux} \doteq m) \wedge (y \doteq q), (\text{aux} \doteq n)\rangle$ (for some variables x, y, domain values p, q and $i, j, m, n \in \mathcal{D}_{\text{aux}}$) in π such that $(x \doteq p)$ and $(y \doteq q)$ are responsible for cost w'. Since π is applicable, a_1 and a_2 are also applicable (in their respective intermediate states). Since intermediate states only differ from $\hat{s}$ in auxiliary and semaphore variables that means that there exist concrete states $s_x, s_y \in \hat{s}$ with $s_x(x) = p$ and $s_y(y) = q$. However, since $\sim_{\text{DD}}$ behaves like $\sim$ for variables of Π and since $w \neq w$' we also have $s'(x) \neq p \vee s'(y) \neq q$ for all $s' \in s^\sim$. This contradicts the requirement on $s^\sim$ being Cartesian, since from $s_x(x) = p$ and $s_y(y) = q$ it follows that there exists a state $s'' \in s^\sim$ with $(x \doteq p) \wedge (y \doteq q) \subseteq s''$. Therefore $w = w'$.

Thus there is a corresponding sequence of transitions in the abstract transition system of Π^{DD} if and only if there is a transition in the abstract transition system of Π. Since every state (and therefore every goal state) s of Π has a corresponding counterpart in Π^{DD} (namely $\text{DD}(s)$) we have that the optimal abstract path costs to a goal state are equal, i.e. $h^{\sim_{\text{DD}}}(\text{DD}(s)) = h^\sim(s)$ for all $s \in \mathcal{S}$ of Π, concluding the proof. □

While this theorem is important for projection abstractions, its importance for Cartesian abstractions is not quite clear, as only some Cartesian abstractions have the non-determinism property. However, we can show another useful property of the EVMDD compilation for arbitrary Cartesian heuristics, which we first show informally by examining once again the abstract transition system depicted in Figure 4.5. Observe that the optimal path between the abstract initial state and the abstract goal state corresponds to the minimization of the transitions induced by the EVMDD compilation. Now, note that this minimization exactly corresponds to the local minimization of the EVMDD given a Cartesian set of states. We have already seen this behaviour in Figure 2.13 when we investigated the relationship between Cartesian sets and the EVMDD compilation (cf. Lemma 6). The following theorem formalizes this observation.

Theorem 16. *Let Π be a planning task and $\sim$ a Cartesian abstraction relation. Let $\sim_{\text{DD}}$ be the corresponding abstraction relation in Π^{DD} which does not aggregate semaphore and auxiliary variables. For each transition $s^\sim \xrightarrow{a,w} s[a]^\sim$ in $\mathcal{T}_\Pi^{\sim_\mathcal{P}}$ there is a sequence of transitions in $\mathcal{T}_{\Pi^{\text{DD}}}^{\sim_\mathcal{P}}$ which starts in $\text{DD}(s)^\sim$, goes to $\text{DD}(s[a])^\sim$ and the cheapest path is exactly $\min_{s' \in s^\sim} c_a(s')$.*

Proof. Follows from Lemma 6. Since $\sim_{\mathrm{DD}}$ preserves auxiliary and semaphore variables we have $(\mathrm{aux} \doteq 0)$, $(\mathrm{lock} \doteq 0) \in \mathrm{DD}(s)^{\sim}$. Furthermore, since $\sim$ is a Cartesian abstraction relation $\mathrm{DD}(s)^{\sim}$ is a Cartesian set, therefore the Lemma is applicable. □

We now have multiple useful properties of Cartesian abstractions. If the Cartesian abstraction does not result in transitions corresponding to non-deterministic actions, then the underlying abstraction heuristic is compilation invariant. But even if there are such non-deterministic transitions, we still get an admissible heuristic value, and depending on the granularity of the Cartesian set where the transition occurs this heuristic might still be very accurate.

4.2.1 CEGAR for tasks with state-dependent action costs

What is left is to discuss *how* to generate Cartesian abstractions. For this, Seipp and Helmert (2018) adapt the counterexample-guided abstraction refinement (CEGAR) algorithm (Clarke et al. 2000) from the model checking community. Intuitively, the CEGAR algorithm starts initially with a very coarse abstraction consisting of a single abstract state. It then iteratively computes a plan for the current abstract transition system, tries to apply the plan to the concrete task, and iteratively refines the abstraction by partitioning an abstract state into finer abstract states such that the reason why the plan was inapplicable vanishes. Algorithm 3, adapted from Seipp and Helmert (2018), describes the CEGAR procedure in detail. In line 2, the initial abstract transition system is built, which just consists of a single abstract state subsuming *all* states. This is easy for classical planning tasks, as we have a single transition for each action $a \in \mathcal{A}$ (possibly representing exponentially many transitions induced by a), but in the presence of state-dependent action costs, a may induce potentially exponentially many transitions with different weights. To solve this problem, we once again underestimate the true transition cost, by setting the weight of a transition induced by a to the cost of a applied in the (single) abstract state ${s_I}^{\sim}$, i.e. the minimum cost of all states subsumed by ${s_I}^{\sim}$. With this, abstract goal distances are still an admissible heuristic, but as a consequence the abstract transition system built by CEGAR is not an abstraction in the sense of Definition 41, at least in regard to transition weights.

The algorithm then proceeds iteratively with its refinement, until some termination condition is satisfied, which usually consists of time and/or memory constraints. In the refinement step, the algorithm first computes a goal *trace*, i.e. a sequence of transitions reaching the goal in the current abstract transition system. If no goal trace is found, then the task is shown to be unsolvable (line 5). Otherwise, the algorithm tries to find a *flaw* in the current trace. For this, the plan induced by the trace is applied to the concrete task Π until either the goal is reached, or the plan is for some reason not applicable to Π. In the first case, we have found a plan for the concrete task and can immediately

return the plan (line 8). Otherwise, the abstract transition system is refined to fix the flaw and the CEGAR algorithm proceeds with its next iteration. The remaining question is what is a flaw and how is the flaw fixed.

Algorithm 3: CEGAR algorithm to generate Cartesian abstractions.

```
1  Function CEGAR()
2      T' = TRIVIALABSTRACTION()
3      while terminal condition is not triggered do
4          τ' = FINDGOALTRACE()
5          if no goal trace found then
6              return task is unsolvable
7          φ = FINDFLAW(τ')
8          if no flaw found then
9              return τ'
10         T' = REFINE(T', φ)
11     return ts'
```

Formally, a flaw $\varphi = \langle s, s_C \rangle$ is a pair of a concrete state s and a Cartesian set $s_C \subseteq s^\sim$, such that applying the abstract plan obtained from τ to Π failed because $s \notin s_C$. We won't give the concrete algorithm sketch to the FINDFLAW function, and instead just repeat the formal description given by Seipp and Helmert (2018). Let $\tau = \langle s'_0 \xrightarrow{a_1,w_1} s'_1, \ldots, s'_{k-1} \xrightarrow{a_k,w_k} s'_k \rangle$, i.e. s'_i is an abstract state. The FINDFLAW algorithm then tries to apply the plan $\pi = \langle a_1, \ldots, a_k \rangle$ to Π, i.e. it tries to find a sequence of transitions $\langle s_0 \xrightarrow{a_1,w_1} s_1, \ldots, s_{k-1} \xrightarrow{a_k,w_k} s_k \rangle$ in $\mathcal{T}_\Pi$ where $s_0 = s_I$, $s_i{}^\sim = s'_i$ for $i \in \{0, \ldots, k\}$ and $s_k \in S_\star$. Starting from the initial state s_0, the next action in π is iteratively applied until one of the following scenarios is encountered.

I *Inapplicability*: a_i is not applicable in the concrete state s_i. The returned flaw is $\langle s_i, s_C \rangle$, where s_C is the set of concrete states in $s_i{}^\sim$ in which a_i is applicable.

II *Successor divergence*: For abstract and concrete transitions $s'_i \xrightarrow{a_i,w_i} s'_{i+1}$ and $s_i \xrightarrow{a_i,w_i} s_{i+1}$ we have that $s_i{}^\sim = s'_i$ but $s_i[a_i]^\sim \neq s'_{i+1}$, i.e. the concrete successor state is not consistent with the abstract successor state. This might happen if the abstract transition system is non-deterministic. Then, the returned flaw is $\langle s_i, s_C \rangle$ where s_C is the set of concrete states in $s_i{}^\sim$ from which s'_{i+1} is reachable by applying a_i.

III *Goal divergence*: The last state s_k is not a goal state. Then, the returned flaw is $\langle s_k, s_C \rangle$ where s_C is the set of concrete goal states in $s^\sim$.

Seipp and Helmert (2018) show that in all these cases s_C is a Cartesian set. If none of these conditions occur and Π is a classical planning task, then π is an optimal plan for Π. However, if Π contains actions with state-dependent costs, then there is an additional potential source for error: the cost implied by an abstract transition does not correspond to the actual cost of applying the action in the concrete transition system. Consider the following example.

Example 22. Consider a planning task Π with two binary variables x, y and two actions $a_1 = \langle \top, (x \doteq 1) \wedge (y \doteq 1) \rangle$ with $c_{a_1} = 2x+1$, and $a_2 = \langle \top, (x \doteq 0) \wedge (y \doteq 1) \rangle$ with $c_{a_1} = 1$. Let $s_I = (x \doteq 1) \wedge (y \doteq 0)$ and $s_\star = (x \doteq 1) \wedge (y \doteq 1)$. Now, consider an abstraction relation $\sim$, such that we have the following abstract states: $s_1^\sim = \{(x \doteq 0) \wedge (y \doteq 0), (x \doteq 1) \wedge (y \doteq 0)\}$, $s_2^\sim = \{(x \doteq 0) \wedge (y \doteq 1)\}$, $s_3^\sim = \{(x \doteq 1) \wedge (y \doteq 1)\}$.

Figure 4.6 depicts the abstract transition system. Since all abstract costs are 1, the optimal abstract plan is $\pi_1 = \langle a_1 \rangle$ with cost 1, and π_1 is also a concrete plan with cost 3. However, the optimal concrete plan is $\pi_2 = \langle a_2, a_1 \rangle$ with concrete and abstract cost 2.

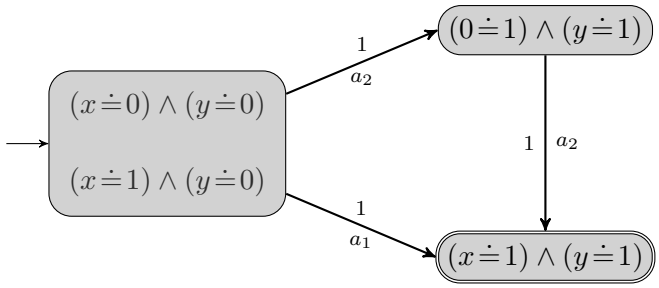


Figure 4.6: Example of an abstract transition system resulting in cost divergence.

We therefore introduce a fourth type of flaw.

IV *Cost divergence:* For abstract and concrete transitions $s'_i \xrightarrow{a_i, w'} s'_{i+1}$ and $s_i \xrightarrow{a_i, w} s_{i+1}$ we have that $s_i{}^\sim = s'_i$ and $s_i[a_i]^\sim = s'_{i+1}$, but $w \neq w'$, i.e. the cost in the abstract transition system is not consistent with the cost in the concrete transition system. Let (i) $v \in \mathrm{vars}(c_{a_i})$ and (ii) let $d, d' \in s^\sim(v)$ be two values that contribute different partial costs in the evaluation of $c_{a_i}(s_i{}^\sim)$ and $d = s(v)$. Then, the returned flaw is $\langle s_i, s_C \rangle$, where $s_C = s_i{}^\sim \setminus \{d\}$. Intuitively, s_C contains states s for which $c_{a_i}(s) \neq w$.

While the first constraint is self-explanatory (it only makes sense to select a variable that influences the action cost), the motivation for the second is best illustrated with another example.

Example 23. Consider an action a with the cost function that is encoded by the EVMDD depicted in Figure 4.7 for two variables x and y with $\mathcal{D}_x = \{0, 1, 2\} = \mathcal{D}_y$, and let $s^\sim$ be the abstract state that is indicated by the highlighted edges. Both x and y satisfy (i), but only x satisfies condition (ii), since both partial costs incurred by y are 2. This can also be observed from the cost of applying a in the contained concrete states, which are $c_a((x \doteq 0) \wedge (y \doteq 1)) = c_a((x \doteq 0) \wedge (y \doteq 2)) = 2$ and $c_a((x \doteq 1) \wedge (y \doteq 1)) = c_a((x \doteq 1) \wedge (y \doteq 2)) = 3$ and hence independent of variable y.

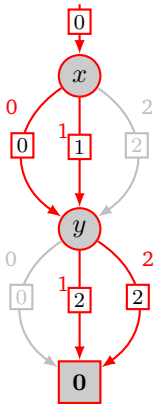


Figure 4.7: EVMDD of Example 23.

The remaining question is now how to refine the abstract transition system (line 10). Given a flaw $\varphi = \langle s, s_C \rangle$, we want to split $s^\sim$ into two abstract states r and t such that φ can not occur in subsequent iterations. Seipp and Helmert (2018) show (Property P6 from Theorem 1) that it is always possible to partition $s^\sim$ into two Cartesian sets that separate s from s_C and they also show that this partitioning is proper, i.e. both subsets (r and t) are proper subsets of $s^\sim$. Thus, in every iteration an abstract state in $\mathcal{T}'$ is split into two new abstract states r and t such that $s \in r$ and $s_C \subseteq t$, and $\mathcal{T}'$ is updated by replacing $s^\sim$ with r and t. What is left is to describe *how to split* $s^\sim$ and *how to update* $\mathcal{T}'$.

For the former, Seipp and Helmert (2018) argue that the only way of splitting $s^\sim$ into two Cartesian sets r and t is to choose a single variable v with $s(v) \notin s_C(v)$ and partition $s^\sim(v)$ into $r(v)$ and $t(v)$, while preserving all other abstract domains. As a consequence, one has to determine how to choose v and how to partition $v(s)$. Based on empirical results, they determine v by choosing a variable which has been refined the most in $s^\sim$, which outperformed two other strategies based on random selection and min-refinement (choosing a variable which has been refined the least), respectively. For the

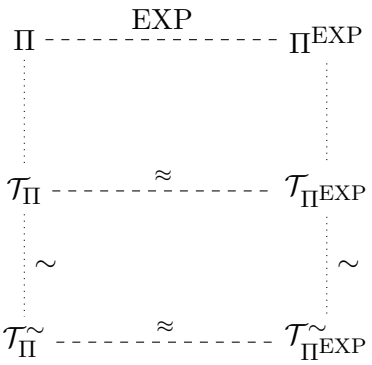
partitioning of $s^\sim(v)$, they decide to put the remaining values of $s^\sim(v) \setminus (\{s(v)\} \cup s_C(v))$ into r, as this is more likely to increase the average heuristic value. In case of cost divergence (IV), we could also select a variable in the support of the cost function which is refined the most. The partitioning, however, is already determined, as $s^\sim(v) \setminus (\{s(v)\} \cup s_C(v)) = \emptyset$, i.e. s_C already contains the remaining domain values.

Finally, we discuss how to update $\mathcal{T}'$. Since only a single abstract state $s^\sim$ is split into two new abstract states r and t, we only have to update incoming and outgoing transitions of $s^\sim$ (including self-loops), by rewiring them to either r or t. In their simple approach, Seipp and Helmert (2018) do this by evaluating for each original transition incoming to (or outgoing from) $s^\sim$ with action a, if there is a corresponding transition for r and t (note that both can happen simultaneously). They also present a more efficient procedure, which only considers transitions induced by the variable v on which the split of $s^\sim$ was based. In our case, the only difference we have are the additional transition weights. However, since the weight w of a transition $s'_i \xrightarrow{a,w'} s'_{i+1}$ is based on the cost of a in s' (i.e. the minimum cost of all states subsumed by s'), determining the *existence* of a transition is not different to the approach by Seipp and Helmert (2018). The weight can then be determined once again by local minimization of the cost function EVMDD based on r and t, respectively. What is left is to update the initial state and the goal state of $\mathcal{T}'$, which is easy, since t can never be the abstract initial state and r can never be the abstract goal state (Seipp and Helmert 2018). This concludes our adaptation of the CEGAR algorithm to tasks with state-dependent action costs.

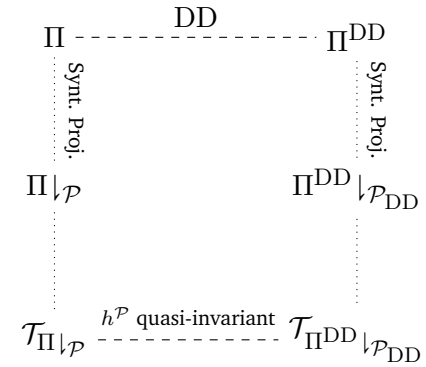
Unfortunately, using a single abstraction obtained from the CEGAR algorithm as the basis for an abstraction heuristic is not competitive with other types of abstraction heuristics, such as iteratively generated PDB heuristics. Therefore, Seipp and Helmert (2018) use *cost partitioning* (Katz and Domshlak 2007, 2010) to come up with different diverse abstractions for which the heuristic estimates can be combined admissibly. Cost partitioning distributes the action cost of the original task over multiple task copies such that the sum of the abstraction heuristics of the task copies is still admissible, and therefore generalizes the canonical PDB heuristic. This led to the introduction of *state-dependent cost partitioning* (Keller et al. 2016) (also called transition cost partitioning in Pommerening 2017), which generalizes cost partitioning, allows cost partitioning to be used for tasks with state-dependent action costs, and also allows to make use of state-dependent action costs for computing heuristics on classical planning tasks. We will discuss this in more detail in chapter 6.

4.3 Summary

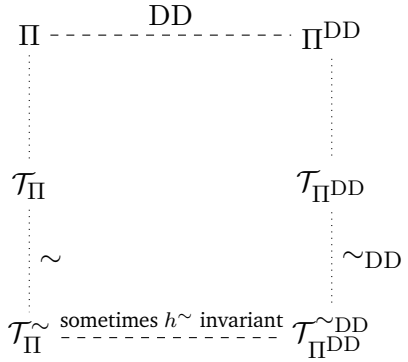
We conclude our theoretical evaluation of abstraction heuristics by summarizing the previous results in Figure 4.8. First, for arbitrary abstraction relations $\sim$, $h^{\sim}$ is invariant under exponential compilation since their abstract transition systems share the same structure (Figure 4.8a). For EVMDD compilation we have different results. EVMDD compilation is invariant for projection heuristics (Figure 4.8b), if we lift the pattern to the auxiliary and semaphore variables. If $\sim$ is Cartesian, then we have $h^{\sim}$ invariance if $\mathcal{T}_{\Pi}^{\sim}$ does not contain non-deterministic transitions and if we preserve auxiliary and semaphore variables. Finally, if there are non-deterministic transitions, then $\mathcal{T}_{\Pi^{\mathrm{DD}}}^{\sim_{\mathrm{DD}}}$ still provides admissible estimates.



(a) Exponential compilation is $h^{\sim}$ invariant.



(b) EVMDD compilation is $h^{\mathcal{P}_{\mathrm{DD}}}$ invariant.



(c) EVMDD compilation result if $\sim$ is Cartesian.

Figure 4.8: Theoretical summary of Chapter 4.

CHAPTER 5

Empirical Evaluation

The theoretical results presented in Chapter 3 and 4 indicate that we can obtain informative heuristics for tasks with state-dependent action costs by considering task compilations. In this section, we are now interested if these theoretical stronger estimates warrant the additional overhead introduced by compilation. For this, we performed different experiments on the benchmark set provided by Speck et al. (2018), which was introduced to compare symbolic planning approaches for tasks with state-dependent action costs. Most domains were adapted from different planning competitions by transforming some actions to actions with state-dependent action costs. In Geißer et al. 2016, another benchmark set based on the determinization of probabilistic planning tasks is provided (i.e. probabilistic effects are replaced with deterministic ones). However, due to the determinization these benchmarks provide no challenge for classical planners. We will therefore focus on the benchmarks provided by Speck et al. (2018) and give a brief introduction of the different domains and highlight the role of the action cost function in each domain.

5.1 Benchmark Set

5.1.1 ASTERIX

In this domain, Asterix must collect the Edelweiss flower, located on top of a mountain. Asterix can climb the mountain and the action cost function of the climb action describes the slope of the mountain. However, Asterix can also choose to first gather a mistletoe and bring it to Getafix who brews him a magic potion. This potion allows Asterix to climb the mountain directly (inducing a cost of 1 in this case). There is one caveat though: to gather a mistletoe Asterix has to ask Obelix to knock out some Romans who hide in the woods. Each knock-out action induces some constant cost. Therefore, for an optimal plan we have to decide if the cost incurred by knocking-out Romans

outweighs the cost of climbing the mountain directly. The tasks differ in the number of Romans and size of the mountain. Figure 5.1 depicts the EVMDD of the climb cost function $(1-p)+(p\cdot(1+(x\cdot(4-y))))$, where x and y specify coordinates of the mountain and p specifies whether the potion was obtained. As there are only few coordinates ($\mathcal{D}_x = \mathcal{D}_y = \{0,1,2\}$) this is one of the easier instances. We point out that this is the only domain that also incorporates actions with conditional effects. As a consequence, our theoretical results do not explicitly hold for this scenario, and some configurations are not able to deal with conditional effects without further modification, even for classical planning tasks.

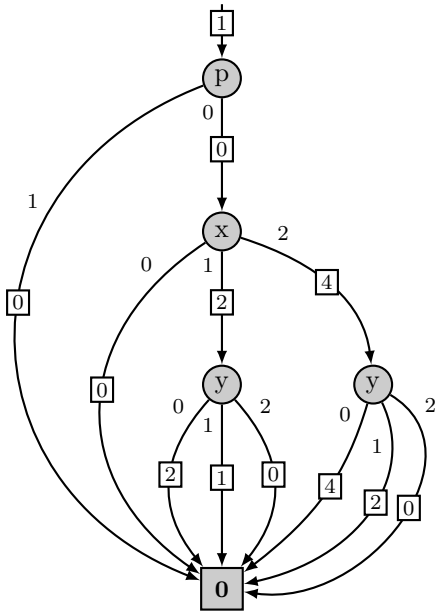


Figure 5.1: EVMDD corresponding to the cost of the climb action in ASTERIX.

5.1.2 GREEDY-PEGSOL

The GREEDY-PEGSOL domain is an adaptation of the classical planning domain PEGSOL, which is based on the Peg Solitaire board game. Here, the goal is to remove all but one peg from the board, by selecting a peg to jump over (and therefore remove) adjacent pegs into a free hole. A single move consists of multiple actions, starting with $\mathrm{begin\text{-}move}$, selecting actions according to the jumps, and ending with $\mathrm{end\text{-}move}$. There are two version of this domain: in the first version, state-dependent costs are induced at the end of a move (i.e. at action $\mathrm{end\text{-}move}$) and the cost of the action is equal to the sum of

pegs left on the board. Therefore an optimal plan prefers to remove as many pegs as possible in the first few moves. In the second version (indicated by GREEDY-PEGSOL2) the state-dependent costs are induced at the beginning of each move. In both versions, the remaining actions (different than $\mathrm{begin\text{-}move}$ and $\mathrm{end\text{-}move}$) induce zero costs. The tasks differ in the arrangement of pegs on the board, but the number of pegs in the domain is always 20 (although in some instances some pegs may already have been removed in the initial state). We won't depict the EVMDD of the cost function here, as this function already contains 20 variables (20 pegs on the board). However, since the cost is just the sum of pegs, the EVMDDs have one non-terminal node for each peg (i.e. 20 non-terminal nodes) and are of similar nature to the EVMDDs in the COLORED GRIPPER domain (although only two domain values per variable).

5.1.3 COLORED GRIPPER

This domain builds upon the well-known GRIPPER domain (McDermott 2000), where a robot equipped with two arms (the grippers) has to rearrange balls from one room to another. In this version, balls and rooms are either colored red or blue. Initially, all balls are located in the blue room, and the goal is to transport all balls to the red room. The cost of moving between rooms is penalized by the number of balls located in a room that does not match the ball color. Picking up a ball induces a cost of zero. Therefore, an optimal plan has to first transport all red balls to the red room, before transporting the blue balls. The tasks differ in the number of balls (but there are always two rooms). Figure 5.2 depicts the EVMDD of the move cost function

$$((2 - b_0) \cdot b_0 \cdot 2) + ((2 - b_1) \cdot b_1 \cdot 2) + |1 - b_2| \cdot (2 - b_2)) + (|1 - b_3| \cdot (2 - b_3)),$$

where b_i specifies if ball i is either in the gripper, in the blue, or in the red room.

5.1.4 OPENSTACKS

The original OPENSTACKS domain (Fink and Voß 1999) is based on a combinatorial optimization problem in which a manufacturer receives different orders, where each order consists of a combination of different items. The manufacturer can only produce one item at a time (but they produce the total quantity required for all orders). An order is *open* as soon as one of its items is produced, and each open order requires a *stack*, representing the temporary storage space for the order. In the original problem, an optimal plan has to arrange production of the items such that the number of open stacks is minimized. In the state-dependent action cost version, the cost of each action is the number of currently open stacks. Therefore, the goal of this version corresponds to minimizing the integral over all open stacks (as opposed to the maximum) (Speck 2018). The tasks differ in the number of items and orders.

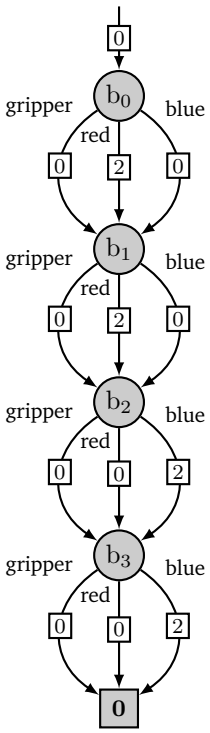


Figure 5.2: EVMDD corresponding to the cost of the move action in COLORED GRIPPER.

Note that since the number of open stacks is encoded as a single counter variable, the action cost function is just the value of this variable. Therefore, the EVMDD representing the action costs consists of a single non-terminal node, with one edge per possible number of open stacks (up to 100 possible open stacks for harder instances).

5.1.5 TRAVELING SALESMAN

In the TRAVELING SALESMAN domain different cities are randomly distributed on a 256×256 grid and the goal is to visit each city exactly one time. The cost of traveling between cities is the Manhattan distance metric and cities are pairwise connected (i.e. every city is reachable from every other city). An optimal plan thus has to find a cheapest path without visiting any city twice. The tasks differ in the number of cities, but not in the grid size. Figure 5.3 depicts parts of the EVMDD of the visit cost function $|x - 70| + |y - 212|$ for a city located at position $(70, 212)$ on the grid. Note that since there are 256 domain values for x and y the EVMDD has a total number of 512 edges.

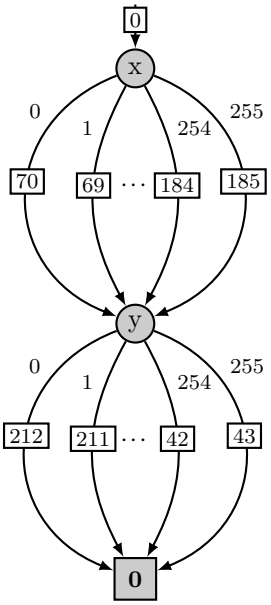


Figure 5.3: EVMDD corresponding to the cost of the visit action in TRAVELING SALESMAN.

5.1.6 Planner setup

While the original (unit/constant) domains are modeled in PDDL, the state-dependent action cost domains are modeled in the Fast Downward (Helmert 2006b) internal input language (cf. Section 2.2.2). The Fast Downward planning system is a state-of-the-art tool to solve classical planning tasks and incorporates many search algorithms and heuristics. For the experiments, the planner was modified such that search algorithms are able to deal with state-dependent action costs by building an EVMDD for each action cost function. Whenever search has to compute the cost of an action for state s, the corresponding EVMDD is evaluated with input s. Note that the underlying Fast Downward version our extension is based on is of March 2017. However, for the heuristics we compare there have not been significant changes since that time.

As we already mentioned at the end of Chapter 2, when we compare heuristics and different compilations we only evaluate the heuristic on the compiled task, but search itself will always search on the original task. Therefore, the way we handle state-dependent action costs in the search is used for all configurations, i.e. even if we use MTMDD or exponential compilation, evaluating state-dependent action costs during search is done by evaluating the corresponding EVMDD. With this, we make sure that our results are only influenced by the performance of the heuristic. That being said, one could also think of other methods to evaluate state-dependent action costs. For ex-

ample, we could parse the action cost functions and transform them to C-like programming functions which are then called during run-time.

To construct different decision diagrams used in the compilation process we rely on different libraries. EVMDDs are constructed and evaluated with the LeMon library (Geißer and Wright 2018), which was initially developed during the work of Mattmüller et al. (2018). Flattened EVMDDs (MTMDDs) are constructed with the Meddly library (Badar and Miner 2011). For both types of diagrams the underlying variable ordering is based on the Fan-In heuristic[1] (Malik et al. 1988), although we note that this only affects the Asterix domain, as the size of the diagrams in the other domains is independent of the ordering. All experiments were conducted with the downward-lab framework (Seipp et al. 2017b) on a cluster of machines based on Intel Xeon E5-2650v2 2.60GHz processors with 64GB DDR3 1866MHz ECC registered memory. Each task was limited to a single core. The time limit was set to 30 minutes and the memory limit to 4GB RAM for all runs. We begin our evaluation by analyzing the different compilations in terms of size and computation time.

5.2 Compilation Results

The compilation of a planning task is computed initially once, and then used for any heuristic computation afterwards. We are interested in the exponential compilation and in different types of EVMDD compilations: EVMDD compilation with quasi-reduced EVMDDs is required to guarantee invariance for the additive heuristic h^{add}. For pattern database heuristics, we have invariance even in the case of fully-reduced EVMDDs. Finally, we require flattened quasi-reduced EVMDDs to guarantee invariance of the maximum heuristic h^{max}.

Two factors are most important for our types of compilation: first, the initial time it requires to produce the compiled task. The more time the compilation process takes, the less time is remaining for search itself. Second, the number of actions generated by the compilation, as this affects the size of the resulting compiled planning task. While the number of variables introduced by compilation is also important, we can always rely on the variable-compact EVMDD compilation to limit the number of additional variables.

We begin by analyzing the number of additional actions. First, quasi-reduced and fully-reduced EVMDD compilation only differ in the number of actions in the ASTERIX domain, in the other domains the size of the EVMDDs does not differ. Figure 5.4 shows that even in ASTERIX, quasi-reduction results at most in a couple of hundred additional actions. In the figure, every point corresponds to an instance. The y-axis depicts the number of actions in the fully-reduced EVMDD compilation, the x-axis depicts the number of actions in the quasi-reduced EVMDD compilation. The diagonal black line serves as a

[1]This ordering heuristic orders the variables according to their depth in the corresponding abstract syntax tree. Variables which appear deeper in the AST are considered more important.

separation to better estimate the differences of the two compilations: points under the line correspond to instances which have less actions in the fully-reduced compilation. Note the logarithmic scale.

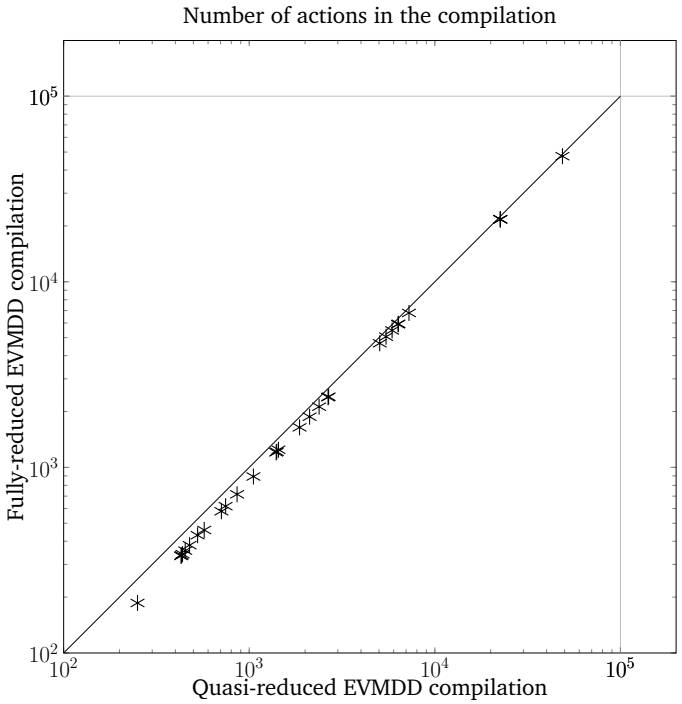


Figure 5.4: Comparison of compiled actions in the Asterix domain between quasi- and fully-reduced EVMDDs.

In Figure 5.5a, we see the comparison between the EVMDD compilation and the flattened EVMDD compilation. Since we require flattened EVMDD compilation for the invariance of the maximum heuristic h^{max}, the flattened EVMDDs have to be quasi-reduced. For ASTERIX and OPENSTACKS, the size of the compilation is around the same, although for the larger instances there is still a difference of a couple of thousand actions. However, for the remaining domains flattened EVMDD compilation produces multiple orders of magnitude more actions than ordinary EVMDD compilation, up to 1.5 million actions for the largest TRAVELING SALESMAN instance. It will therefore be interesting if the additional information in the maximum heuristic outweighs the large increase in the task size. Note that we have only two points per GREEDY-PEGSOL domain, since most instances use the same cost function.

Interestingly, for some domains we have a similar picture when we compare the EVMDD compilation with the exponential compilation, depicted in Figure 5.5b. For OPENSTACKS, the exponential compilation results in marginally fewer actions than EVMDD compilation. Since the cost function depends only on a single variable, EVMDD compilation introduces initial and conclud-

ing actions, which are not required in the exponential compilation. Interestingly, for all TRAVELING SALESMAN and most ASTERIX instances, exponential compilation is still feasible. Only for the four largest ASTERIX instances the compilation could not be computed, due to reaching the memory limit of 4GB. For COLORED GRIPPER, exponential compilation was only feasible in 6 out of 31 instances, and for both types of the GREEDY-PEGSOL domain compilation was not possible for any instance, as even the simplest instances have 20 pegs and therefore 2^{20} different valuations of the action cost function. Note that while the border is indicated with 10^7, the line indicates that these compilations were not completed.

Finally, we compare the time it takes to produce the compiled task. For both types of EVMDD compilation (quasi-reduced and fully-reduced), compile time takes most often less than 0.1 seconds, only for some of the larger COLORED GRIPPER domains the time increases to up to 0.5 seconds[2]. We therefore do not visualize these results. Figure 5.6 depicts the time required for exponential and MTMDD compilation. As we can see, for most instances the time required is less than a second, only for some of the TRAVELING SALESMAN instances it takes a few seconds to compile the task. Furthermore, we can say that while some tasks are forgiving for the exponential compilation (namely TRAVELING SALESMAN and OPENSTACKS) other tasks (GREEDY-PEGSOL and COLORED GRIPPER) are infeasible.

To conclude the analysis of the different compilations, we briefly mention another type of compilation, which we can use as a baseline for the different compilations. The $\min$ compilation transforms all actions with state-dependent action costs to constant cost actions, by setting the cost of an action to the minimum possible cost the function can take on for any state. This is efficiently possible since the minimum cost value corresponds to the input weight of the corresponding EVMDD. Obviously, this compilation does not increase task size, but greatly underestimates action costs. Nevertheless, if an optimal plan mostly consists of action applications which minimize action cost, then this might still be an informative compilation. It will now be interesting to see how the different compilations perform under different heuristics.

5.3 Search Results

5.3.1 Establishing coverage bounds

Our initial experiment conducts uninformed $A^\star$ search, where we set the h-value to the cheapest minimal cost of any applicable action ("blind search"). Note that this still performs $A^\star$ search on a task with state-dependent action

[2]While ASTERIX and OPENSTACKS have much more actions, compilation time is usually dominated by the time it takes to construct the decision diagram.

costs (by evaluating the corresponding cost EVMDD during search). With this baseline we were already able to solve 144 out of 256 instances (Table 5.1, first column), where over a half of the solved instances are due to the GREEDY-PEGSOL domains. We therefore already have a lower bound on the number of tasks we can solve. To establish some form of upper bound on the coverage (number of solved instances), we also evaluate $A^\star$ performance with the additive heuristic instead of blind search. As the additive heuristic is not admissible, the resulting plans are *not optimal*, but since it is easier to find any plan than an optimal plan we can use this result as an upper bound. We evaluated three different compilations: the baseline compilation $\min$, the exponential compilation EXP and the EVMDD compilation DD with quasi-reduced EVMDDs (to guarantee invariance). We also evaluated flattened EVMDD compilation, but since this compilation does not have any advantage for h^{add} it was strictly dominated by the non-flattened EVMDD compilation. Table 5.1 shows the results for coverage, best results are highlighted in bold font. As exponential compilation is not feasible for both GREEDY-PEGSOL domains and many ASTERIX instances, this compilation does not perform well overall. However, in OPENSTACKS and TRAVELING SALESMAN it performs exceptionally well. For OPENSTACKS this is as expected, as exponential compilation has even less overhead than EVMDD compilation. But even for TRAVELING SALESMAN the compilation solves all instances, despite generating over a million actions in the hardest instances. The $\min$ compilation does not present much benefit, given that GREEDY-PEGSOL is already nearly fully solved with blind $A^\star$. Nevertheless, the number of solved OPENSTACKS instances is interesting, given that all actions in this domain have state-dependent costs, and the minimum cost of each action is 0. Finally, EVMDD compilation performs overall the best, which coincides with our theoretical results for the additive heuristic.

To increase the upper bound, we also conducted a greedy best-first search (GBFS) with the additive heuristic. Greedy best-first search is similar to $A^\star$, but states are expanded by only comparing their heuristic estimate (whereas $A^\star$ expands states with minimal f-value). Table 5.2 shows the coverage for the different configurations. For the exponential compilation we do not get any different results than before. For the $\min$ compilation, however, *all* of the ASTERIX instances are solved. This can be explained by the fact that in this compilation the climb action gets assigned a cost of 1, and the greedy behaviour thus always prefers to climb the mountain immediately. Examining the plan cost confirms this conjecture. In an instance with 10 Romans and a mountain size of 75×75, the plan returned by GBFS with $\min$ compilation has a cost of 514904, while the plan returned by DD compilation has a cost of 169 (which is also the cost of an optimal plan). Besides ASTERIX, the coverage is similar to the $A^\star$ results for the different compilations. Interestingly, coverage of the COLORED GRIPPER domain does not increase, even if we only require non-optimal plans. Finally, we mention that we also evaluated the maximum heuristic and PDB heuristics with GBFS, but they were strictly dominated by

Coverage	$A^\star_{\text{blind}}$	$A^{\star\min}_{h^{add}}$	$A^{\star\text{EXP}}_{h^{add}}$	$A^{\star\text{DD}}_{h^{add}}$
ASTERIX (30)	**10**	9	5	**10**
GREEDY-PEGSOL (50)	48	48	0	**50**
GREEDY-PEGSOL2 (50)	**48**	**48**	0	**48**
COLORED GRIPPER (30)	8	8	4	**9**
OPENSTACKS (70)	9	32	**46**	**46**
TRAVELING SALESMAN (26)	21	21	**26**	**26**
Sum (256)	144	166	81	**189**

Table 5.1: Coverage for $A^\star$ with blind and additive heuristic.

Coverage	$\text{GBFS}^{\min}_{h^{add}}$	$\text{GBFS}^{\text{EXP}}_{h^{add}}$	$\text{GBFS}^{\text{DD}}_{h^{add}}$
ASTERIX (30)	**30**	5	9
GREEDY-PEGSOL (50)	44	0	**50**
GREEDY-PEGSOL2 (50)	**50**	0	**50**
COLORED GRIPPER (30)	8	4	**10**
OPENSTACKS (70)	34	**46**	**46**
TRAVELING SALESMAN (26)	21	**26**	**26**
Sum (256)	187	81	**191**

Table 5.2: Coverage for GBFS with additive heuristic.

the additive heuristic.

While these experiments show that the compilation based approach for the additive heuristic pays off, we note that this evaluation is mostly aimed at getting a rough estimate of how many tasks we should expect to be able to solve optimally. There are many different techniques for satisficing planning (see for example the planner abstracts of the latest International Planning Competition (Torralba and Pommerening 2018)), which would most likely solve even more tasks.

5.3.2 Evaluating the maximum heuristic

We now discuss our results for the maximum heuristic. From our theoretical results in Chapter 3 we know that this heuristic is only invariant under exponential and under flattened EVMDD compilation. Nevertheless, we have seen (Figure 5.5a) that the number of actions in the GREEDY-PEGSOL, TRAVELING SALESMAN and COLORED GRIPPER domains is much higher than with EVMDD compilation. This raises the question if the additional guidance provided by the heuristic outweighs the increase in compilation size (which increases the

time required to compute the heuristic). Guidance is often measured in terms of number of *node expansions*, i.e. states expanded by the $A^\star$ algorithm, as more accurate heuristic estimates most often *reduce* the number of expanded nodes. Figure 5.7a compares the number of expansions between the normal and the flattened EVMDD compilation. Interestingly, the flattened EVMDD compilation only marginally reduces the number of expanded nodes, suggesting a marginal increase in heuristic guidance. We also evaluated the expansions of the exponential compilation, which in theory should be equal to the flattened EVMDD compilation, as both yield the same heuristic estimate. Indeed, if the exponential compilation was possible, the number of expansions is exactly the same. Having seen that flattened EVMDD compilation only marginally increases the guidance, we are now interested how this compilation impacts the planning time. Figure 5.7b shows the total planning time for both EVMDD compilations, i.e. the total time in seconds it requires the planner to find a plan, including precomputation time (such as compilation time). Data points for which one configuration reached the time or memory limit are omitted. As we can see, the total planning time is *always* higher for flattened EVMDD compilation, for TRAVELING SALESMAN and GREEDY-PEGSOL even by multiple orders of magnitude. This coincides with the increase in compilation size shown in Figure 5.5a.

Before we investigate how this all affects coverage, we first compare the $\min$ compilation with non-flattened EVMDD compilation. Figure 5.8a and 5.8b shows again the number of node expansions and the total search time (again, omitting data points in which one configuration reached the time or memory limit), this time comparing the $\min$ compilation with the quasi-reduced EVMDD compilation. We can see that the approximation of the $\min$ compilation hurts heuristic accuracy, but mostly in the GREEDY-PEGSOL domain. However, this does not pay off in terms of total planning time, which is almost always less when we use the $\min$ compilation. Given these results, we would expect that the $\min$ compilation performs best when applying the maximum heuristic. Table 5.3 indeed shows that out of all compilations, the $\min$ compilation performs the best. Even worse, the other compilations solve less tasks than blind search. Therefore, at least for these benchmarks our more sophisticated compilations do not pay off when we consider the maximum heuristic.

5.3.3 Evaluating PDB heuristics

We now evaluate the Canonical heuristic and the iterative Pattern Database approach (h^{iPDB}) by Haslum et al. (2007), for which we use the implementation in Fast Downward, as described by Sievers et al. (2012) (cf. the later part of Section 4.1). Since this heuristic does not support conditional effects we exclude the ASTERIX domain from our results.

For PDB heuristics, the theoretical results of Chapter 4 allow us to con-

Coverage	blind	h^{max}_{min}	h^{max}_{DD}	h^{max}_{FDD}
ASTERIX (30)	**10**	**10**	8	5
GREEDY-PEGSOL (50)	**48**	**48**	**48**	42
GREEDY-PEGSOL2 (50)	**48**	**48**	46	42
COLORED GRIPPER (30)	**8**	**8**	**8**	**8**
OPENSTACKS (70)	9	**10**	**10**	**10**
TRAVELING SALESMAN (26)	**21**	**21**	20	15
Sum (256)	144	**145**	140	122

Table 5.3: Coverage for $A^\star$ with the maximum heuristic.

sider fully-reduced EVMDDs instead of quasi-reduced EVMDDs. Furthermore, to guarantee invariance we have to adapt the generated patterns to include semaphore and auxiliary variables. Therefore, we evaluate different configurations regarding h^{iPDB}: first, we use h^{iPDB} with the variable-compact EVMDD compilation (here still denoted as DD) without any adaptation of the algorithm. As a result, the canonical heuristic will not be invariant. Then, we adapt the h^{iPDB} algorithm as described in Section 4.1: initially, there is a pattern for each goal variable except for the semaphore and the auxiliary variable. These are added to the initial patterns. Then, the test for additivity is modified, such that semaphore and auxiliary variable are additive with all other variables. Finally, when we sample states up to a specific depth and the state is an intermediate state (i.e. semaphore variable is not zero) we apply the chain of intermediate actions until a non-intermediate state is retrieved. We already note that this affects tasks with many variables in the support of the action cost function more, since we have to generate more actions until a non-intermediate state is reached. For the parameters of h^{iPDB} we use the standard configuration of Fast Downward: the maximum size of the pattern collections can go up to 20 million abstract states (with up to 2 million abstract states per pattern), for each new candidate collection we sample 1000 states for which we try to improve the heuristic value in at least 10 states (otherwise the candidate is discarded). We also do not enable any time limit, i.e. as long as the previous mentioned limits are not reached we continue with pattern generation in the precomputation phase.

Once again, Figure 5.9a depicts the number of expansions, now for the original and the adapted iterative PDB heuristic. While the adapted heuristic is most of the time more informative, this is not always the case (note the two TRAVELING SALESMAN instances), although in these cases the planning tasks are really easy. If we compare the total planning time (Figure 5.10a), we see that if the support includes many variables (as it is the case in GREEDY-PEGSOL), the total planning time rises. To see that this is indeed due to the longer precomputation time we also include the pure search time, i.e. after the

heuristic was already precomputed, depicted in Figure 5.10b. As we can see, search time for both GREEDY-PEGSOL domains is often equal, sometimes even faster with the adapted configuration. Unfortunately, the long precomputation time results in many of the harder GREEDY-PEGSOL instances not being solved, due to reaching the time limit.

To see how accurate these heuristic estimates are we also compare the number of expanded nodes with the exponential compilation (where possible). While both heuristics are invariant if they include the same patterns (lifted to auxiliary and semaphore variables), the difference during pattern search in the sampling of states (based on average action cost) might result in different estimates. Interestingly, comparing the expansions of the adapted heuristic on EVMDD compilation with the original heuristic on the exponential compilation (Figure 5.9b), we see that for TRAVELING SALESMAN the exponential compilation is much more informative. Upon further investigation, this result is due to the difference in the number of patterns: for TRAVELING SALESMAN the exponential compilation results in 5-8 more patterns in the larger tasks, compared to the adapted heuristic on the EVMDD compilation. This might be due to the fact that the average action cost of the exponential compilation is two times higher than that of EVMDD compilation. Therefore, both configurations sample states at different depths. Additionally, if the heuristic value of the initial state is not being improved in 10 sampled states, then pattern generation stops early. This is also the reason for the outlier in one of the OPENSTACKS instances: pattern generation in the exponential compilation stopped immediately, as the initial pattern collection was already good enough for almost all sampled states, while for the EVMDD compilation at least 10 sampled states were found where the heuristic value can be improved. Indeed, if we compare the total planning time between both configurations (Figure 5.11), the adapted configuration often requires less total time, since the pattern generation finishes much earlier, except for the harder instances where the more informed heuristic outweighs the additional precomputation time.

We now evaluate the coverage of different h^{iPDB} configurations. Table 5.4 depicts the results for different configurations: blind search, original h^{iPDB} with the $\min$ compilation, original h^{iPDB} with the exponential compilation, original h^{iPDB} with the EVMDD compilation, and the adapted iPDB heuristic with the EVMDD compilation. While overall the blind heuristic outperforms the other approaches, this is due to the GREEDY-PEGSOL domain, where the precomputation phase often takes up several hundreds of seconds (or even over thousand seconds in the adapted approach), since we set no precomputation time limit. Interestingly, in TRAVELING SALESMAN, the exponential compilation is able to solve one additional task. This is an important result, as it indicates that the theoretical result of invariance of PDB heuristics can be used in such tasks, but only if we are able to direct the search for pattern collections in a similar way.

Coverage	blind	$h^{\text{iPDB}}_{\text{min}}$	$h^{\text{iPDB}}_{\text{EXP}}$	$h^{\text{iPDB}}_{\text{DD}}$	adapted $h^{\text{iPDB}}_{\text{DD}}$
GREEDY-PEGSOL (50)	**48**	44	0	44	41
GREEDY-PEGSOL2 (50)	**48**	38	0	44	37
COLORED GRIPPER (30)	**8**	**8**	5	**8**	**8**
OPENSTACKS (70)	9	**10**	**10**	**10**	**10**
TRAVELING SALESMAN (26)	21	22	**23**	22	22
Sum (226)	**134**	122	38	128	112

Table 5.4: Coverage for different h^{iPDB} configurations.

Finally, we note some other configurations we tested. We additionally evaluated h^{iPDB} EVMDD compilation with no semaphore. Interestingly, the number of expansions did not differ to original h^{iPDB} on the EVMDD compilation with semaphore, suggesting that at some point most collections will include the semaphore variable anyway. We also evaluated the non variable-compact EVMDD compilation. The results match with the intuition that the additional number of variables negatively impacts the heuristic behaviour and therefore this configuration often already reaches the memory limit at the precomputation phase. Finally, we tried to increase the depth where states are sampled. This resulted in much more accurate heuristic estimates for the OPENSTACKS domain, where the original depth estimate (influenced by the average action cost) was much lower than it is in reality. However, the increased depth also increased the precomputation time, and in the end this configuration did not pay off in terms of coverage.

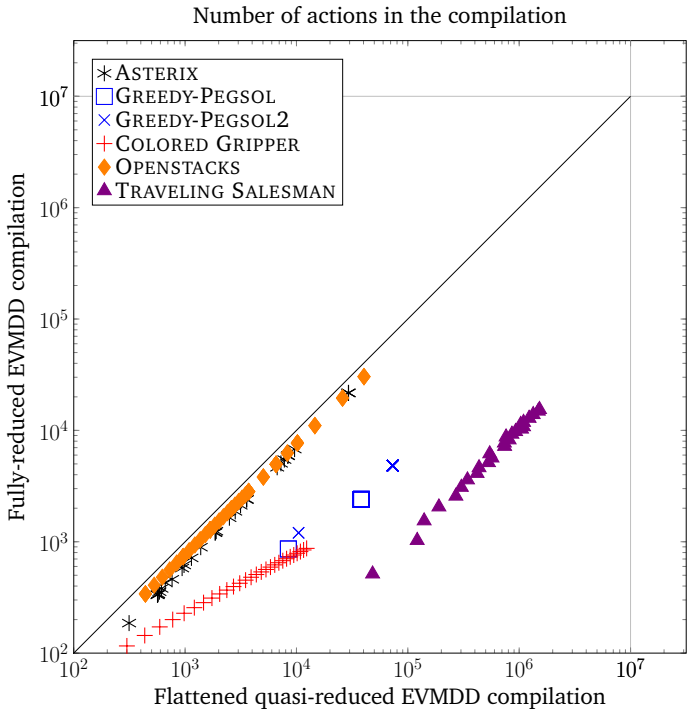
5.4 Discussion

We conclude the empirical evaluation with a brief discussion of the previous results. First, as the benchmark set was introduced to compare symbolic planning approaches some domains are not well-suited for heuristic forward search. For example, for classical planning the GRIPPER domain is known to be very hard for standard $A^\star$ search: Helmert and Röger (2008) show that if the heuristic is only off by a cost of 1, then about half of all reachable states need to be considered. If the heuristic is off by 2, then $A^\star$ corresponds to a breadth-first search with duplicate elimination. On the other hand, symbolic approaches perform very well in this domain (Torralba 2015), as well as in OPENSTACKS.

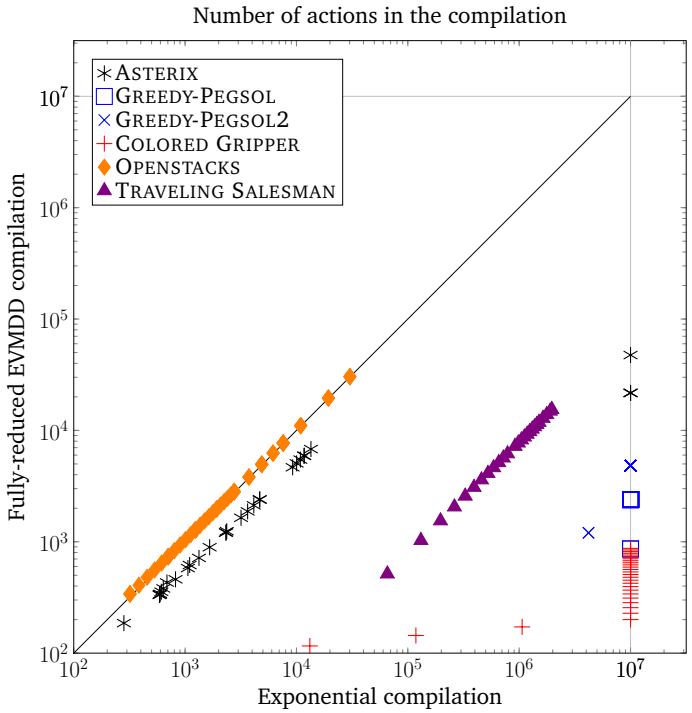
For GREEDY-PEGSOL, the heuristics provide not much additional information, the original h^{iPDB} heuristic yields for all compilations exactly the same amount of node expansions in this domain, and the adapted heuristic is only marginally better (we do not depict this here, as nearly all data entries lie

on the diagonal line). If heuristic performance does not pay off anyway, then heuristic invariance does not matter.

In contrast to this, in TRAVELING SALESMAN the exponential compilation shows that a good heuristic helps to solve more tasks. While with EVMDD compilation h^{iPDB} does not solve more instances than blind search, the harder instances are solved in significantly less time (instance 22 requires 635 seconds of total planning time with blind search, while adapted h^{iPDB} requires only 267 seconds). This suggests the following: first, in tasks where state-dependent action costs play an important role (and can't merely be underestimated by the minimum cost) applying EVMDD compilation can be beneficial. Second, while PDB heuristics are invariant under EVMDD compilation, the way the pattern collection is generated is *not* invariant under EVMDD compilation. This warrants a further look in the adaptation of the h^{iPDB} heuristic, which we leave up for future work.



(a) Comparison of compiled actions between flattened EVMDD and normal EVMDD compilation.



(b) Comparison of compiled actions between exponential and EVMDD compilation.

Figure 5.5

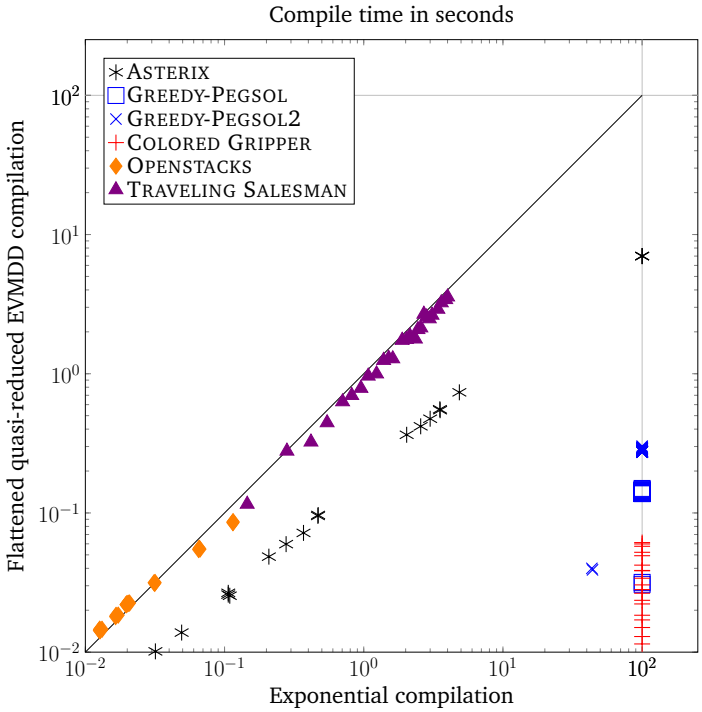
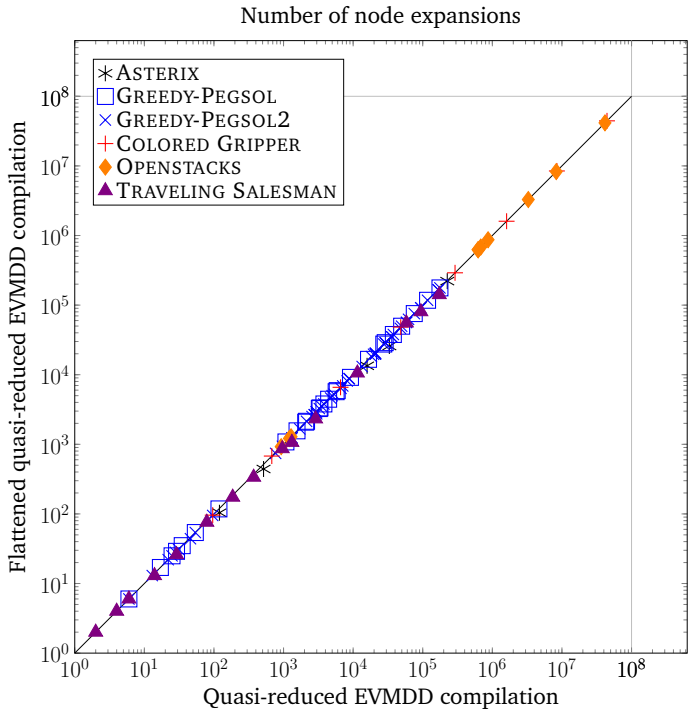
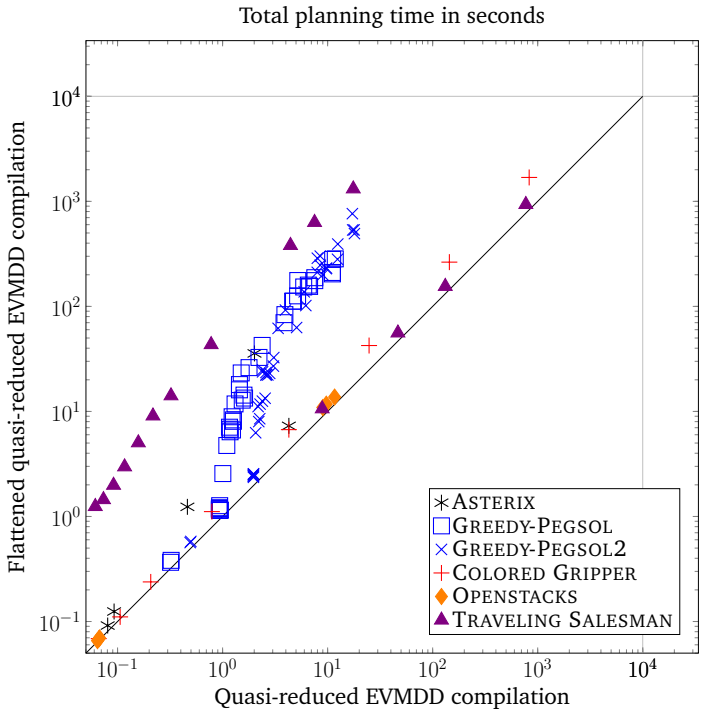


Figure 5.6: Comparison of compile time between exponential and flattened EVMDD compilation.

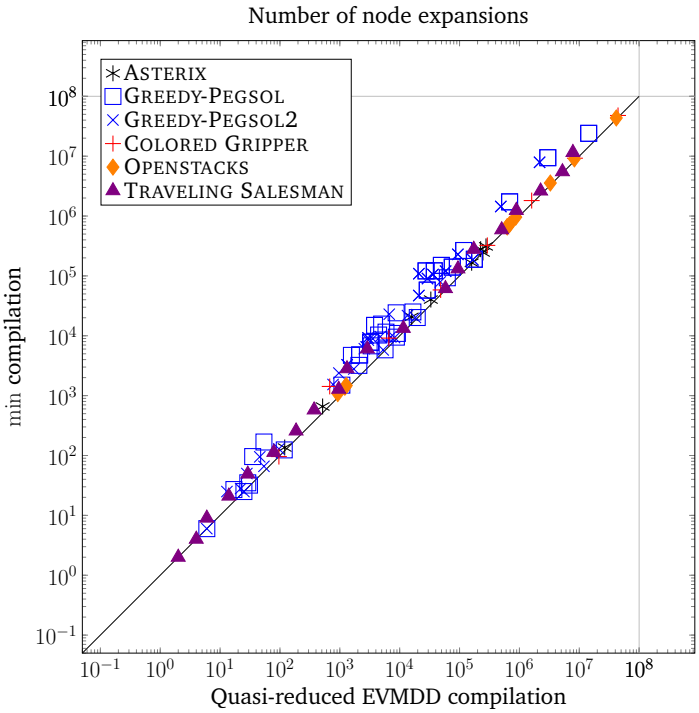


(a) Comparison of node expansions with h^{max} between normal and flattened EVMDD compilation.

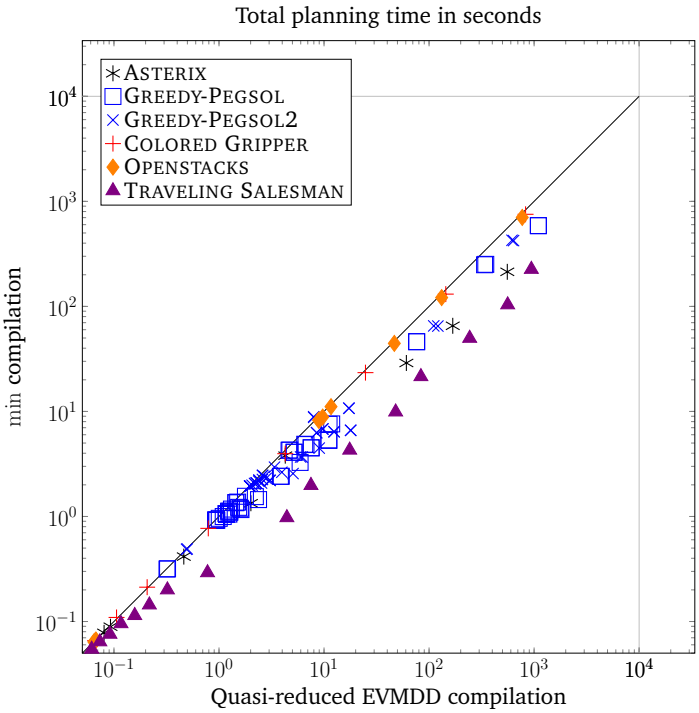


(b) Comparison of total planning time in seconds with h^{max} between normal and flattened EVMDD compilation.

Figure 5.7

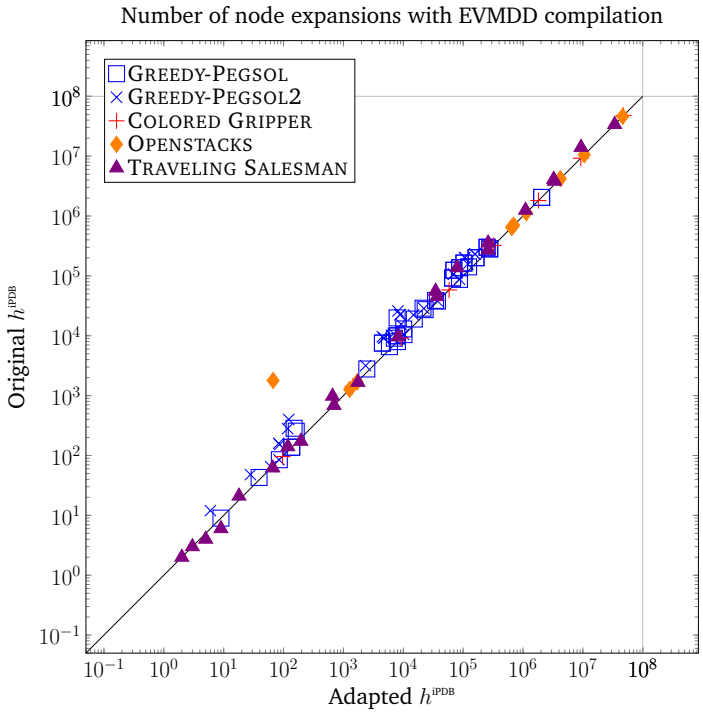


(a) Comparison of node expansions with h^{max} between $\min$ compilation and EVMDD compilation.

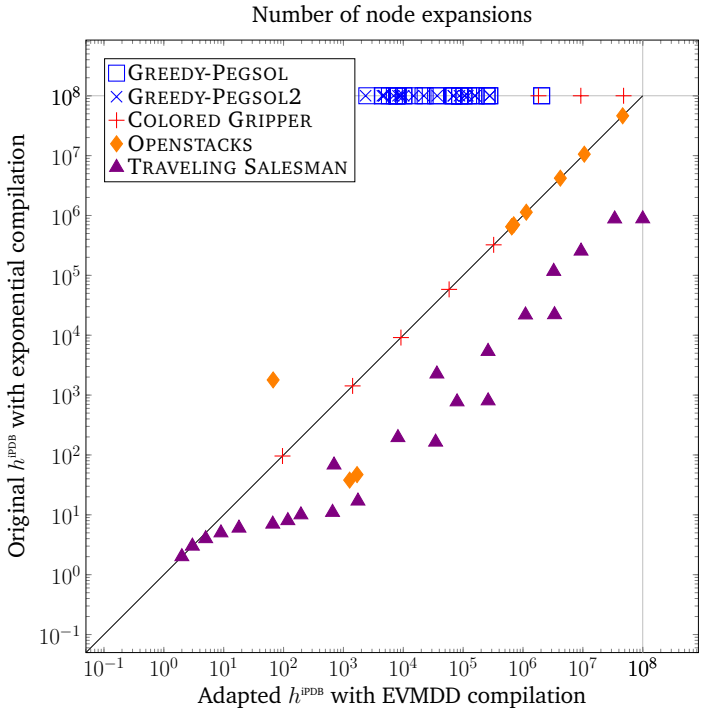


(b) Comparison of total planning time in seconds with h^{max} between $\min$ compilation and EVMDD compilation.

Figure 5.8

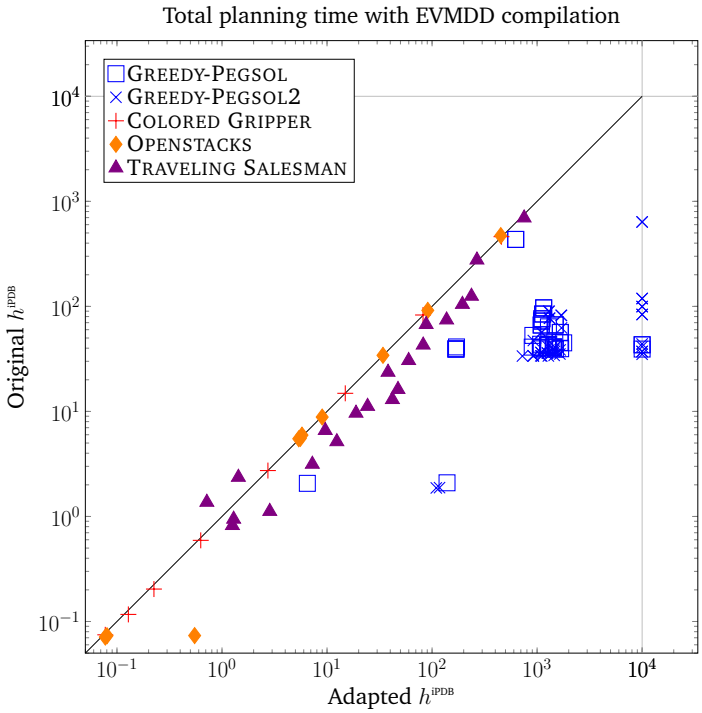


(a) Comparison of node expansions between the original and the adapted h^{iPDB} heuristic with EVMDD compilation.

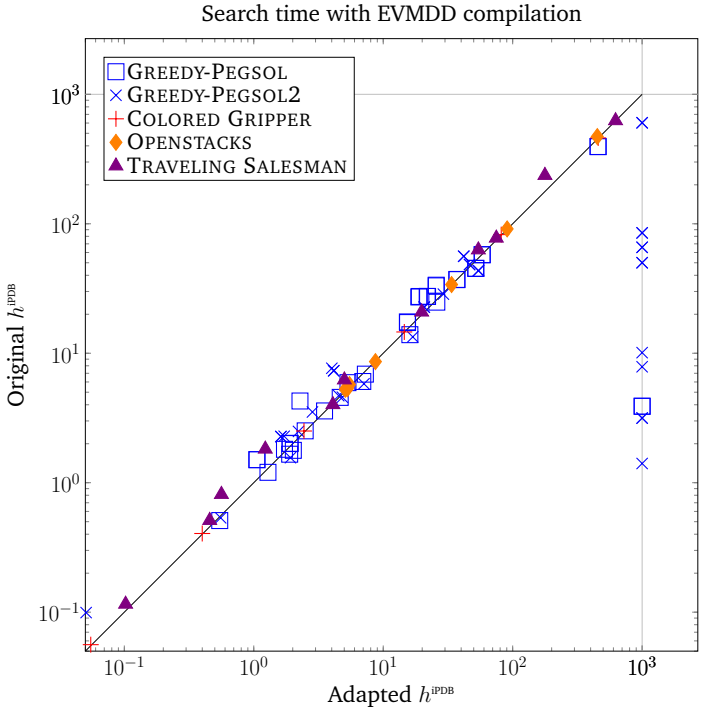


(b) Comparison of node expansions between the original h^{iPDB} heuristic based on exponential compilation and the adapted h^{iPDB} heuristic based on EVMDD compilation.

Figure 5.9



(a) Comparison of total planning time in seconds between the original and the adapted h^{iPDB} heuristic with EVMDD compilation.



(b) Comparison of search time in seconds between the original and the adapted h^{iPDB} heuristic with EVMDD compilation.

Figure 5.10

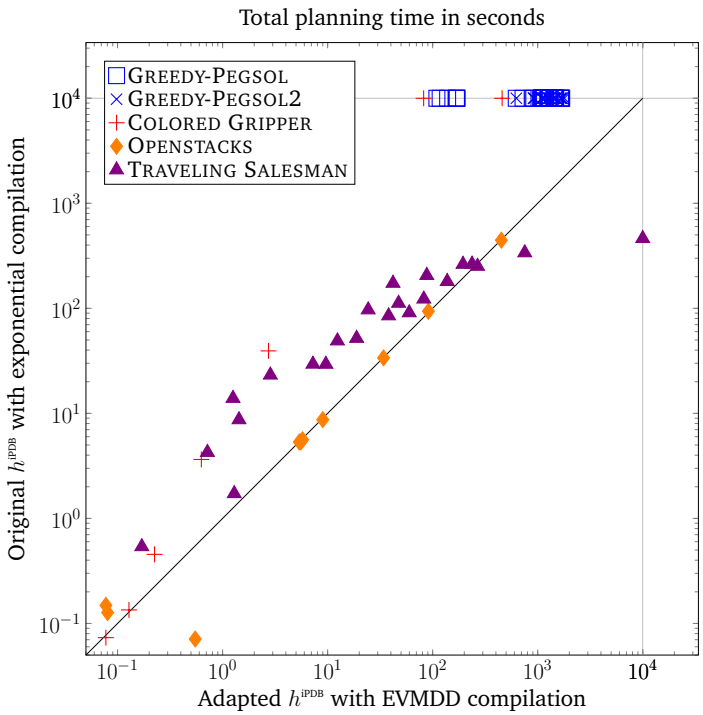


Figure 5.11: Comparison of total planning time in seconds between the original h^{iPDB} heuristic based on exponential compilation and the adapted h^{iPDB} heuristic based on EVMDD compilation.

CHAPTER 6

Further Reading and Future Work

We now discuss some possible future work and give further references related to the different topics we discussed previously.

6.1 Delete Relaxation

One of the pitfalls of delete relaxation heuristics is their inability to reflect the consumption of resources or the movement between locations. For example, in the relaxation of the Logistics domain the truck is simultaneously at multiple locations and packages can always be considered as not loaded in the truck. Therefore, Katz et al. (2013) propose to only relax a part of the problem and they dub their approach *red-black planning*. In red-black planning, variables may have two types of semantics: the red variables take the relaxed semantics, i.e. they accumulate their values instead of switching between them. Black variables, on the other hand, take the usual semantics of switching between values. This allows to reflect the consumption of a particular resource, while still relaxing the remaining parts of the problem. If the black variables fulfill some requirements (namely the dependencies between black variables are acyclic, and each black variable is *invertible*), then red-black planning is tractable (Domshlak et al. 2015). This technique is most commonly used in satisficing planners (Torralba and Pommerening 2018).

Since it is a classical planning technique we can apply it to the EVMDD compiled task. The question is, once again, if red-black planning heuristics are invariant under compilation. We conjecture that, similar to abstractions, red-black planning is invariant if semaphore and auxiliary variables are preserved, i.e. painted black. Since red-black planning techniques (e.g. the Mercury planner, Katz and Hoffmann 2014) automatically paint variables black by analyzing the causal graph (Helmert 2006b), we would have to analyze if this would include the semaphore and auxiliary variables. If this is not the case, we could adapt the algorithm to automatically include these variables,

similar to our adaptation of the hill-climbing algorithm for PDBs.

6.2 Cartesian Abstractions and Cost Partitioning

We did not include Cartesian abstraction heuristics in our empirical evaluation, as for a competitive performance multiple abstraction heuristics have to be combined. This is possible with cost partitioning (Katz and Domshlak 2007, 2010), where the original action cost is distributed over multiple copies of the planning task. Then, an abstraction is generated for each copy, and the abstract heuristic values may be admissibly combined. *General cost partitioning* (Pommerening et al. 2015) generalizes cost partitioning by allowing negative costs, which leads to even more accurate heuristic estimates. While optimal cost partitioning can be computed in polynomial time (Katz and Domshlak 2007, 2010), it can be expensive to compute even a single optimal cost partitioning for explicit abstractions with reasonable size (Pommerening et al. 2013; Seipp et al. 2017a).

Saturated cost partitioning (Seipp and Helmert 2014, 2018) is an approximation of an optimal cost partitioning. Given an ordered sequence of heuristics and an overall cost function, the saturated cost partitioning algorithm iteratively computes the minimum cost required to preserve the heuristic and uses the remaining cost as the new overall cost function. Seipp and Helmert (2018) show how to interweave saturated cost partitioning with the CEGAR algorithm, to iteratively compute abstraction heuristics based on the current overall cost function. As a result, the different abstraction heuristics may be summed up admissibly, often leading to much stronger heuristic estimates (Seipp and Helmert 2018).

In Keller et al. (2016), we generalize cost partitioning even further, by introducing general *state-dependent* cost partitioning (also called transition cost partitioning in Pommerening 2017). This allows to distribute the action cost differently for different states and does not only enable cost partitioning for tasks with state-dependent action costs, but an *optimal general state-dependent cost partitioning* also dominates the state-independent version for classical planning tasks. We present *saturated state-dependent cost partitioning*, which allows a more fine-grained distribution of the action costs among the different abstractions. While the state-dependent version of saturated cost partitioning is incomparable to its state-independent version (there exist examples which show that neither dominates the other) it sometimes surpasses optimal state-independent cost partitioning. This work also makes use of EVMDDs to represent state-dependent cost functions, and shows how to efficiently use this representation to distribute the action costs over the different Cartesian abstractions.

In the work of Keller et al. (2016) we use saturated state-dependent cost partitioning as a tool for classical planning tasks, but since it already incorpo-

rates state-dependent action costs it can be immediately applied to tasks with state-dependent action costs. One drawback of the approach is the increased fractioning of the action costs and as a result the EVMDD representation results in an exponential blow-up in some domains. Possible future work is to take a look into dynamic variable ordering approaches, to reduce the size of the EVMDDs during construction of the different abstractions. Another possible approach would be to take a look into the EVMDD compilation and compare the behaviour between saturated state-independent cost partitioning on the compiled task and saturated state-dependent cost partitioning on the original task.

6.3 Conditional Costs and Conditional Effects

While this thesis mostly ignored conditional effects, in Mattmüller et al. 2018 we show that conditional effects and conditional (state-dependent) costs are closely related: considering them separately can lead to less informed heuristics. We therefore introduce a generalization of EVMDDs to functions defined over monoids, which allows to represent conditional effects and state-dependent action costs at the same time by combining them in a single EVMDD. This EVMDD can be used to compute relaxation heuristics and Bergdoll (2018) shows that Cartesian abstraction heuristics (with non-deterministic abstract transitions) and delete relaxation heuristics are invariant under a compilation based on the combined EVMDD (i.e. which compiles costs *and* effects away). As this includes projection heuristics, this brings up an interesting question: can we compile conditional effects away while preserving PDB heuristics, in the same way we did with the compilation of state-dependent costs? If so, we could use this compilation to enable the canonical heuristic (and therefore h^{iPDB}) on tasks with conditional effects.

6.4 General Action Cost Functions

In this thesis we only considered action cost functions which resulted in positive costs for *all* states. This is in line with classical tasks having only actions with positive cost, but we could also think of an action cost function which has positive costs in all *reachable* states. This gives the modeler more room in modeling their tasks, as they must not be concerned with costs for unreachable states. For example, if we know that in a 2D coordinate system only the part where $x > y$ is reachable, we can express costs as $x - y$, which results in a cost of -1 for state $(x \doteq 0) \wedge (y \doteq 1)$. In general, the techniques in this thesis are also applicable in this case, but our theoretical results regarding invariance under EVMDD compilation change. For example, since in such a case the EVMDD representing the action cost function has an incoming edge with negative weight (since the minimum of the function is negative), the resulting

EVMDD compilation generates an action with negative costs (the initial action a^{pre}). Therefore, for delete relaxation we quickly end up with a heuristic value of $-\infty$ (due to infinite application of a^{pre}), even if the relaxed state only includes states for which we have positive action costs. This is not the case for the heuristic estimate in the original task, therefore invariance under EVMDD compilation is not guaranteed in this case. We conjecture that we can still preserve invariance if we use flattened EVMDDs. Since these only induce costs in concluding actions, and since the cost is only negative if the corresponding EVMDD path corresponds to a state inducing negative costs, the relaxation heuristic should only produce a value of $-\infty$ if the relaxed state contains such a state with negative action cost. In this case, the heuristic on the original task would also yield an estimate of $-\infty$.

Furthermore, it would also be interesting to examine decision diagrams based on other types of expansion, for example Functional Decision Diagrams (Kebschull et al. 1992) based on the Davio-expansion or Kronecker Functional Decision Diagrams (KFDDs), which are based on both Shannon and Davio expansion. This could result in a compilation which is exponentially more compact for the type of functions that result in compact KFDDs. However, while the additive nature of EVMDDs makes them suitable for compilation, it is not obvious how to do this for decision diagrams with other function evaluation mechanisms.

6.5 Probabilistic Planning

Finally, we mention that the results given in this thesis also open up the application of classical planning methods to probabilistic planning. Many probabilistic planning tasks are given as factored Markov Decision Processes (MDPs) (Puterman 1994) or Stochastic Shortest Path problems (SSPs) (Bertsekas and Tsitsiklis 1996), where the costs (or rewards in the case of MDPs) are state-dependent. A common approach to approximate a solution for such probabilistic tasks (which can be used as a heuristic to guide the probabilistic search) is to solve a determinized version of the task (Yoon et al. 2007, 2008), i.e. transform the problem to a deterministic planning problem. However, if the problem includes state-dependent costs, then previously these also had to be approximated. Our results allow the use of classical state-of-the-art planners for such determinization approaches, without further approximation of action costs. In Geißer et al. (2015) and Geißer et al. (2016) we used such an approach to improve the heuristic underlying the probabilistic planning system PROST (Keller and Eyerich 2012), used to initialize the state-value function.

While our research focuses on the state-dependent costs of such problems, Trevizan et al. (2017) developed *occupation measure heuristics* for stochastic shortest path problems (SSPs). Occupation measures were the first domain-independent heuristic that reasons about probabilities, and they can be seen

as a generalization of operator-counting heuristics (Pommerening et al. 2014), which is a cost-partitioning heuristic. Therefore, there currently co-exist cost-partitioning heuristics that deal with probabilistic effects, as well as cost-partitioning heuristics that deal with state-dependent action costs. Combining both most likely leads to a powerful tool for finding solutions to probabilistic planning tasks.

CHAPTER 7

Conclusion

We introduced classical planning with state-dependent action costs, which allows to model naturally arising cost functions in planning tasks, instead of passing the load of the management of such functions on the modeler. By representing action cost functions as edge-valued multi-valued decision diagrams we can exhibit structure present in the cost function, which allows us to use the decision diagram to efficiently compute the minimum cost for Cartesian states, which includes relaxed states, as well as abstract states in Cartesian abstractions. The representation as decision diagrams also allowed us to introduce several compilations of tasks with state-dependent action costs. Unlike the exponential compilation, the resulting compilation is compact if the underlying decision diagram is also compact.

We generalized delete relaxation heuristics and abstraction heuristics to tasks with state-dependent action costs and our result for efficient minimization over Cartesian states enables us to efficiently compute this generalized version. We have also given multiple results about the invariance of such heuristics under different compilations. This allows us to apply the additive heuristic or the maximum heuristic (in case of flattened decision diagrams) on the compiled task, without forfeiting heuristic accuracy. By adapting the pattern collection underlying the canonical heuristic we also achieved invariance for this heuristic under all types of the presented compilations. Furthermore, we discussed how we can adapt the CEGAR algorithm, which allows the generation of arbitrary Cartesian abstraction heuristics for tasks with state-dependent action costs.

Our empirical evaluation shows that depending on the type of the problem it may be sufficient to underapproximate the costs without relying on more sophisticated techniques. However, if the state-dependent nature of the problem plays a major role, then the compilation with decision diagrams is useful and reduces search effort.

List of Figures

List of Tables

List of Algorithms

Bibliography

Akers Jr., Sheldon B. (1978). “Binary Decision Diagrams”. In: *IEEE Transactions on Computers* 27.6, pp. 509–516.

Aldinger, Johannes and Bernhard Nebel (2017). “Interval Based Relaxation Heuristics for Numeric Planning with Action Costs”. In: *Proceedings of the 40th Annual German Conference on Artificial Intelligence (KI 2017)*. Ed. by Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm. Vol. 10505. Lecture Notes in Artificial Intelligence. Springer-Verlag, pp. 15–28.

Bäckström, Christer and Bernhard Nebel (1995). “Complexity Results for SAS^+ Planning”. In: *Computational Intelligence* 11.4, pp. 625–655.

Badar, Junaid and Andrew Miner (2011). *MEDDLY: Multi-terminal and Edge-valued Decision Diagram LibrarY*. Accessed: 2018-10-08. URL: `http://meddly.sourceforge.net/`.

Bahar, R. Iris, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi (1993). “Algebraic Decision Diagrams and Their Applications”. In: *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1993)*. Ed. by Michael R. Lightner and Jochen A. G. Jess, pp. 188–191.

— (1997). “Algebraic Decision Diagrams and Their Applications”. In: *Formal Methods in System Design* 10.2–3, pp. 171–206.

Baier, Jorge A., Fahiem Bacchus, and Sheila A. McIlraith (2007). “A Heuristic Search Approach to Planning with Temporally Extended Preferences”. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*. Ed. by Manuela M. Veloso, pp. 1808–1815.

Ball, Thomas, Andreas Podelski, and Sriram K. Rajamani (2001). “Boolean and Cartesian Abstraction for Model Checking C Programs”. In: *Proceed-*

ings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001), pp. 268–283.

Bast, Hannah, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck (2016). "Route Planning in Transportation Networks". In: Lecture Notes in Computer Science 9220, pp. 19–80.

Becker, Bernd and Rolf Drechsler (1995). "How many decomposition types do we need?" In: *1995 European Design and Test Conference, ED&TC 1995, Paris, France, March 6-9, 1995*, pp. 438–443.

Becker, Bernd, Rolf Drechsler, and Ralph Werchner (1995). "On the Relation between BDDs and FDDs". In: *Information and Computation* 123.2, pp. 185–197.

Bergdoll, Rolf-David (2018). "EVMDD-based Action Compilations: Dealing with Auxiliary Variables in Abstraction Heuristics". MA thesis. University of Freiburg.

Bergman, David, André Augusto Ciré, Willem-Jan van Hoeve, and John N. Hooker (2016). "Discrete Optimization with Decision Diagrams". In: *INFORMS Journal on Computing* 28.1, pp. 47–66.

Bertsekas, Dimitri P. and John N. Tsitsiklis (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bollig, Beate, Martin Löbbing, and Ingo Wegener (1995). "Simulated Annealing to improve variable orderings for BDDs". In: *Workshop Notes of the International Workshop on Logic Synthesis (IWLS 1995)*.

Bollig, Beate and Ingo Wegener (1996). "Improving the Variable Ordering of OBDDs Is NP-Complete". In: *IEEE Transactions on Computers* 45.9, pp. 993–1002.

Bonet, Blai and Héctor Geffner (1999). "Planning as Heuristic Search: New Results". In: *Recent Advances in AI Planning. 5th European Conference on Planning (ECP 1999)*. Ed. by Susanne Biundo and Maria Fox. Vol. 1809. Lecture Notes in Artificial Intelligence. Heidelberg: Springer-Verlag, pp. 360–372.

Bonet, Blai, Gábor Loerincs, and Héctor Geffner (1997). "A Robust and Fast Action Selection Mechanism for Planning". In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*. AAAI Press, pp. 714–719.

Boole, George (1854). *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Dover Publications.

Brace, Karl S., Richard Rudell, and Randal E. Bryant (1990). "Efficient Implementation of a BDD Package". In: *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC 1990)*. Ed. by Richard C. Smith, pp. 40–45.

Bryant, Randal E. (1986). "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* 35.8, pp. 677–691.

Bylander, Tom (1994). "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 69.1–2, pp. 165–204.

Ceriani, Luca and Alfonso E. Gerevini (2015). "Planning with Always Preferences by Compilation into STRIPS with Action Costs". In: *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SoCS 2015)*. Ed. by Levi Lelis and Roni Stern. AAAI Press, pp. 161–165.

Ciardo, Gianfranco and Radu Siminiceanu (2002). "Using Edge-Valued Decision Diagrams for Symbolic Generation of Shortest Paths". In: *Proceedings of the Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD 2002)*. Ed. by Mark Aagaard and John W. O'Leary. Vol. 2517. Lecture Notes in Computer Science. Springer-Verlag, pp. 256–273.

Clarke, Edmund M., Masahiro Fujita, Patrick C. McGeer, Kenneth L. McMillain, Jerry Chih-Yuan Yang, and Xudong Zhao (1993a). "Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation". In: *Workshop Notes of the International Workshop on Logic Synthesis (IWLS 1993)*.

Clarke, Edmund M., Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith (2000). "Counterexample-Guided Abstraction Refinement". In: *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*. Ed. by E. Allen Emerson and A. Prasad Sistla, pp. 154–169.

Clarke, Edmund M., Orna Grumberg, and Doron A. Peled (1999). *Model Checking*. The MIT Press.

Clarke, Edmund M., Kenneth L. McMillan, Xudong Zhao, Masahiro Fujita, and Jerry Chih-Yuan Yang (1993b). "Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping". In: *Proceedings of the 30th Design Automation Conference (DAC 1993)*. Ed. by Alfred E. Dunlop, pp. 54–60.

Codognet, Philippe and Daniel Diaz (1996). "A Simple and Efficient Boolean Solver for Constraint Logic Programming". In: *Journal of Automated Reasoning* 17.1, pp. 97–128.

Culberson, Joseph C. and Jonathan Schaeffer (1998). "Pattern Databases". In: *Computational Intelligence* 14.3, pp. 318–334.

Cutland, Nigel J. (1980). *Computability — An Introduction to Recursive Function Theory*. Cambridge University Press.

Dijkstra, Edsger W. (1959). "A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1, pp. 269–271.

Domshlak, Carmel, Jörg Hoffmann, and Michael Katz (2015). "Red-black planning: A new systematic approach to partial delete relaxation". In: *Artificial Intelligence* 221, pp. 73–114.

Dräger, Klaus, Bernd Finkbeiner, and Andreas Podelski (2009). "Directed model checking with distance-preserving abstractions". In: *International Journal on Software Tools for Technology Transfer* 11.1, pp. 27–37.

Drechsler, Rolf and Bernd Becker (1998). "Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions". In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 17.10, pp. 965–973.

Drechsler, Rolf, Bernd Becker, and Nicole Göckel (1995). "A Genetic Algorithm for Variable Ordering of OBDDs". In: *Workshop Notes of the International Workshop on Logic Synthesis (IWLS 1995)*, pp. 55–64.

Edelkamp, Stefan (2001). "Planning with Pattern Databases". In: *Proceedings of the Sixth European Conference on Planning (ECP 2001)*. Ed. by Amedeo Cesta and Daniel Borrajo. AAAI Press, pp. 84–90.

Edelkamp, Stefan and Malte Helmert (2001). "The Model Checking Integrated Planning System (MIPS)". In: *AI Magazine* 22.3, pp. 67–71.

Edelkamp, Stefan and Peter Kissmann (2009). "Optimal Symbolic Planning with Action Costs and Preferences". In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*. Ed. by Craig Boutilier. AAAI Press, pp. 1690–1695.

— (2011). "On the Complexity of BDDs for State Space Search: A Case Study in Connect Four". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, pp. 18–23.

Edelkamp, Stefan, Peter Kissmann, and Álvaro Torralba (2015). "BDDs Strike Back (in AI Planning)". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, pp. 4320–4321.

Edelkamp, Stefan and Frank Reffel (1998). "OBDDs in Heuristic Search". In: *Proceedings of the 22nd Annual German Conference on Artificial Intelligence (KI 1998)*. Ed. by Otthein Herzog and Andreas Günter. Vol. 1504. Lecture Notes in Computer Science. Springer-Verlag, pp. 81–92.

Eyerich, Patrick, Robert Mattmüller, and Gabriele Röger (2009). "Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning". In: *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*. Ed. by Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis. AAAI Press, pp. 130–137.

Fast Downward website (2018). *Fast Downward documentation - evaluators*. Accessed: 2018-10-08. URL: `http://www.fast-downward.org/Doc/Evaluator#Canonical_PDB`.

Fink, Andreas and Stefan Voß (1999). "Applications of modern heuristic search methods to pattern sequencing problems". In: *Computers & OR* 26.1, pp. 17–34.

Fox, Maria and Derek Long (2003). "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20, pp. 61–124.

Gazen, B. Cenk and Craig A. Knoblock (1997). "Combining the Expressivity of UCPOP with the Efficiency of Graphplan". In: *Recent Advances in AI Planning. 4th European Conference on Planning (ECP 1997)*. Ed. by Sam Steel and Rachid Alami. Vol. 1348. Lecture Notes in Artificial Intelligence. Springer-Verlag, pp. 221–233.

Geißer, Florian, Thomas Keller, and Robert Mattmüller (2014). "Past, Present, and Future: An Optimal Online Algorithm for Single-Player GDL-II Games." In: *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. Ed. by Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan. IOS Press, pp. 357–362.

— (2015). "Delete Relaxations for Planning with State-Dependent Action Costs". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. Ed. by Qiang Yang and Michael Wooldridge. AAAI Press, pp. 1573–1579.

— (2016). "Abstractions for Planning with State-Dependent Action Costs". In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*. Ed. by Amanda Coles, Andrew Coles, Stefan Edelkamp, Daniele Magazzeni, and Scott Sanner. AAAI Press, pp. 140–148.

Geißer, Florian and Benedict Wright (2018). *LEMon-DD: Library for Edge-valued Monoid-based Decision Diagrams*. Accessed: 2019-03-04. URL: `https://github.com/geisserf/lemon-dd`.

Gerevini, Alfonso E. and Derek Long (2005). *Plan Constraints and Preferences in PDDL3*. Tech. rep. R. T. 2005-08-47. University of Brescia, Department of Electronics for Automation.

Hansen, Eric A., Rong Zhou, and Zhengzhu Feng (2002). "Symbolic Heuristic Search Using Decision Diagrams". In: *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*. Ed. by Sven Koenig and Robert C. Holte. Vol. 2371. Lecture Notes in Artificial Intelligence. Springer-Verlag, pp. 83–98.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Haslum, Patrik, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig (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)*. AAAI Press, pp. 1007–1012.

Helmert, Malte (2002). "Decidability and Undecidability Results for Planning with Numerical State Variables". In: *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*. Ed. by Malik Ghallab, Joachim Hertzberg, and Paolo Traverso. AAAI Press, pp. 303–312.

— (2003). "Complexity results for standard benchmark domains in planning". In: *Artificial Intelligence* 143.2, pp. 219–262.

— (2006a). "New Complexity Results for Classical Planning Benchmarks". In: *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*. Ed. by Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey. AAAI Press, pp. 52–61.

— (2006b). "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.

— (2009). "Concise Finite-Domain Representations for PDDL Planning Tasks". In: *Artificial Intelligence* 173, pp. 503–535.

Helmert, Malte, Patrik Haslum, Jörg Hoffmann, and Raz Nissim (2014). "Merge-and-Shrink Abstraction: A Method for Generating Lower Bounds in Factored State Spaces". In: *Journal of the ACM* 61.3, 16:1–63.

Helmert, Malte and Hauke Lasinger (2010). "The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem". In: *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*. Ed. by Ronen Brafman, Héctor Geffner, Jörg Hoffmann, and Henry Kautz. AAAI Press, pp. 234–237.

Helmert, Malte, Florian Pommerening, and Gabriele Röger (2017). *Lecture on Planning and Optimization*. Accessed: 2018-10-08. URL: https://ai.dmi.unibas.ch/_files/teaching/hs17/po/exercises/sheet04.pdf.

Helmert, Malte and Gabriele Röger (2008). "How Good is Almost Perfect?" In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*. AAAI Press, pp. 944–949.

Helmert, Malte, Gabriele Röger, and Silvan Sievers (2015). "On the Expressive Power of Non-Linear Merge-and-Shrink Representations". In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*. Ed. by Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein. AAAI Press, pp. 106–114.

Hoey, Jesse, Robert St-Aubin, Alan Hu, and Craig Boutilier (1999). "SPUDD: Stochastic Planning using Decision Diagrams". In: *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI 1999)*, pp. 279–288.

Hoffmann, Jörg (2015). "Simulated Penetration Testing: From Dijkstra to Turing Test++". In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*. Ed. by Ronen Brafman, Carmel Domshlak, Patrik Haslum, and Shlomo Zilberstein. AAAI Press, pp. 364–372.

Hoffmann, Jörg and Bernhard Nebel (2001). "The FF Planning System: Fast Plan Generation Through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14, pp. 253–302.

Holte, Robert C. (2010). "Common Misconceptions Concerning Heuristic Search". In: *Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS 2010)*. Ed. by Ariel Felner and Nathan Sturtevant. AAAI Press, pp. 46–51.

Hooker, John N. (2013). "Decision Diagrams and Dynamic Programming". In: *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013)*. Ed. by Carla Gomes and Meinolf Sellmann. Springer-Verlag, pp. 94–110.

ICAPS competitions (2018). *ICAPS competitions*. Accessed: 2018-09-10. URL: http://www.icaps-conference.org/index.php/Main/Competitions.

Ivankovic, Franc, Patrik Haslum, Sylvie Thiébaux, Vikas Shivashankar, and Dana S. Nau (2014). "Optimal Planning with Global Numerical State Constraints". In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*. Ed. by Steve Chien, Alan Fern, Wheeler Ruml, and Minh Do. AAAI Press, pp. 145–153.

Iverson, Kenneth E. (1962). “A programming language”. In: *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference (AIEE-IRE 62)*. Vol. 21. ACM New York, pp. 345–351.

Katz, Michael and Carmel Domshlak (2007). “Structural Patterns of Tractable Sequentially-Optimal Planning”. In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*. Ed. by Mark Boddy, Maria Fox, and Sylvie Thiébaux. AAAI Press, pp. 200–207.

— (2010). “Optimal admissible composition of abstraction heuristics”. In: *Artificial Intelligence* 174.12–13, pp. 767–798.

Katz, Michael and Jörg Hoffmann (2014). “Mercury Planner: Pushing the Limits of Partial Delete Relaxation”. In: *Eighth International Planning Competition (IPC-8): planner abstracts*, pp. 43–47.

Katz, Michael, Jörg Hoffmann, and Carmel Domshlak (2013). “Red-Black Relaxed Plan Heuristics”. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*. Ed. by Marie desJardins and Michael L. Littman. AAAI Press, pp. 489–495.

Kebschull, Udo, Endric Schubert, and Wolfgang Rosenstiel (1992). “Multilevel Logic Synthesis Based on Functional Decision Diagrams”. In: *Proceedings of the 29th ACM/IEEE Design Automation Conference (DAC 1992)*. Ed. by Daniel G. Schweikert, pp. 43–47.

Keller, Thomas and Patrick Eyerich (2012). “PROST: Probabilistic Planning Based on UCT”. In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. Ed. by Lee McCluskey, Brian Williams, José Reinaldo Silva, and Blai Bonet. AAAI Press, pp. 119–127.

Keller, Thomas and Florian Geißer (2015). “Better Be Lucky Than Good: Exceeding Expectations in MDP Evaluation”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, pp. 3540–3547.

Keller, Thomas, Florian Pommerening, Jendrik Seipp, Florian Geißer, and Robert Mattmüller (2016). “State-dependent Cost Partitionings for Cartesian Abstractions in Classical Planning”. In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*. Ed. by Subbarao Kambhampati. AAAI Press, pp. 3161–3169.

Keyder, Emil and Héctor Geffner (2008). “Heuristics for Planning with Action Costs Revisited”. In: *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008)*, pp. 588–592.

Korf, Richard E. (1997). “Finding Optimal Solutions to Rubik’s Cube Using Pattern Databases”. In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI 1997)*. AAAI Press, pp. 700–705.

Korf, Richard E., Michael Reid, and Stefan Edelkamp (2001). “Time complexity of iterative-deepening A*”. In: *Artificial Intelligence* 129, pp. 199–218.

Lai, Yung-Te (1993). “Logic Verification and Synthesis using Function Graphs”. PhD thesis. University of Southern California.

Lai, Yung-Te, Massoud Pedram, and Sarma B. K. Vrudhula (1996). “Formal Verification Using Edge-Valued Binary Decision Diagrams”. In: *IEEE Transactions on Computers* 45.2, pp. 247–255.

Lai, Yung-Te and Sarma Sastry (1992). “Edge-Valued Binary Decision Diagrams for Multi-Level Hierarchical Verification”. In: *Proceedings of the 29th ACM/IEEE Design Automation Conference (DAC 1992)*. Ed. by Daniel G. Schweikert, pp. 608–613.

Lee, Chang-Yeong (1959). “Representation of Switching Circuits by Binary-Decision Programs”. In: *Bell System Technical Journal* 38.4, pp. 985–999.

Malik, Sharad, Albert R. Wang, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli (1988). “Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment”. In: *Proceedings of the 1988 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1988)*, pp. 6–9.

Masoumi, Arman, Megan Antoniazzi, and Mikhail Soutchanski (2015). “Modeling Organic Chemistry and Planning Organic Synthesis”. In: *Global Conference on Artificial Intelligence, GCAI 2015*, pp. 176–195.

Mattmüller, Robert, Florian Geißer, Benedict Wright, and Bernhard Nebel (2017). “On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning”. In: *ICAPS 2017 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*, pp. 10–18.

— (2018). “On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*. AAAI Press, pp. 6237–6245.

McDermott, Drew (1996). “A Heuristic Estimator for Means-Ends Analysis in Planning”. In: *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS 1996)*. Ed. by Brian Drabble. AAAI Press, pp. 142–149.

— (2000). “The 1998 AI Planning Systems Competition”. In: *AI Magazine* 21.2, pp. 35–55.

McDermott, Drew, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins (1998). *PDDL – The Planning Domain Definition Language – Version 1.2*. Tech. rep. CVC TR-98-003/DCS TR-1165. Yale University: Yale Center for Computational Vision and Control.

Miller, D. Michael (1993). "Multiple-Valued Logic Design Tools". In: *23rd IEEE International Symposium on Multiple-Valued Logic (ISMVL 1993)*. IEEE Computer Society, pp. 2–11.

Mirkis, Vitaly and Carmel Domshlak (2007). "Cost-Sharing Approximations for h^+". In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*. Ed. by Mark Boddy, Maria Fox, and Sylvie Thiébaux. AAAI Press, pp. 240–247.

Muise, Christian (2015). *Planning.Domains - A collection of tools for working with planning domains*. Accessed: 2018-09-11. URL: `http://www.planning.domains`.

Muller, David E. (1954). "Application of Boolean algebra to Switching Circuit Design and to Error Detection". In: *Trans. I.R.E. Prof. Group on Electronic Computers* 3.3, pp. 6–12.

Orkin, Jeff (2006). "Three states and a plan: the AI of F.E.A.R". In: *Proceedings of the Game Developers Conference (GDC)*.

Panda, Shipra and Fabio Somenzi (1995). "Who are the variables in your neighborhood". In: *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1995)*. Ed. by Richard L. Rudell, pp. 74–77.

Petrick, Ronald and Mary Ellen Foster (2013). "Planning for Social Interaction in a Robot Bartender Domain". In: *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*. Ed. by Daniel Borrajo, Subbarao Kambhampati, Angelo Oddi, and Simone Fratini. AAAI Press, pp. 389–397.

Pommerening, Florian (2017). "New Perspectives on Cost Partitioning for Optimal Classical Planning". PhD thesis. University of Basel.

Pommerening, Florian, Malte Helmert, Gabriele Röger, and Jendrik Seipp (2015). "From Non-Negative to General Operator Cost Partitioning". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*. AAAI Press, pp. 3335–3341.

Pommerening, Florian, Gabriele Röger, and Malte Helmert (2013). "Getting the Most Out of Pattern Databases for Classical Planning". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*. Ed. by Francesca Rossi. AAAI Press, pp. 2357–2364.

Pommerening, Florian, Gabriele Röger, Malte Helmert, and Blai Bonet (2014). "LP-based Heuristics for Cost-optimal Planning". In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*. Ed. by Steve Chien, Alan Fern, Wheeler Ruml, and Minh Do. AAAI Press, pp. 226–234.

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Reed, Irving S. (1954). "A class of multiple-error-correcting codes and the decoding scheme". In: *Trans. of the IRE Professional Group on Information Theory (TIT)* 4, pp. 38–49.

Rice, Michael and Sanjay Kulhari (2008). *A survey of static variable ordering heuristics for efficient BDD/MDD construction*. Tech. rep. Available at `http://alumni.cs.ucr.edu/~skulhari/StaticHeuristics.pdf`. University of California.

Röger, Gabriele, Florian Pommerening, and Malte Helmert (2014). "Optimal Planning in the Presence of Conditional Effects: Extending LM-Cut with Context Splitting". In: *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. Ed. by Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan. IOS Press, pp. 765–770.

Rudell, Richard (1993). "Dynamic Variable Ordering for Ordered Binary Decision Diagrams". In: *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1993)*. Ed. by Michael R. Lightner and Jochen A. G. Jess, pp. 42–47.

Russel, Stuart J. and Peter Norvig (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

Sanner, Scott (2010). *Relational Dynamic Influence Diagram Language (RDDL): Language Description*.

Sanner, Scott and Davic McAllester (2005). "Affine Algebraic Decision Diagrams (AADDs) and their Application to Structured Probabilistic Inference". In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Professional Book Center, pp. 1384–1390.

Scala, Enrico, Patrik Haslum, Daniele Magazzeni, and Sylvie Thiébaux (2017). "Landmarks for Numeric Planning Problems". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*. Ed. by Carles Sierra. AAAI Press, pp. 4384–4390.

Scherrer, Sascha, Florian Pommerening, and Martin Wehrle (2015). "Improved Pattern Selection for PDB Heuristics in Classical Planning (Extended Abstract)". In: *Proceedings of the Eighth Annual Symposium on Combina-*

torial Search (SoCS 2015). Ed. by Levi Lelis and Roni Stern. AAAI Press, pp. 216–217.

Seipp, Jendrik and Malte Helmert (2013). "Counterexample-guided Cartesian Abstraction Refinement". In: *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*. Ed. by Daniel Borrajo, Subbarao Kambhampati, Angelo Oddi, and Simone Fratini. AAAI Press, pp. 347–351.

— (2014). "Diverse and Additive Cartesian Abstraction Heuristics". In: *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*. Ed. by Steve Chien, Alan Fern, Wheeler Ruml, and Minh Do. AAAI Press, pp. 289–297.

— (2018). "Counterexample-Guided Cartesian Abstraction Refinement for Classical Planning". In: *Journal of Artificial Intelligence Research* 62, pp. 535–577.

Seipp, Jendrik, Thomas Keller, and Malte Helmert (2017a). "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)*. AAAI Press, pp. 3651–3657.

Seipp, Jendrik, Florian Pommerening, Silvan Sievers, and Malte Helmert (2017b). *Downward Lab*. `https://doi.org/10.5281/zenodo.790461`. DOI: `10.5281/zenodo.790461`. URL: `https://doi.org/10.5281/zenodo.790461`.

Sievers, Silvan, Manuela Ortlieb, and Malte Helmert (2012). "Efficient Implementation of Pattern Database Heuristics for Classical Planning". In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*. Ed. by Daniel Borrajo, Ariel Felner, Richard Korf, Maxim Likhachev, Carlos Linares López, Wheeler Ruml, and Nathan Sturtevant. AAAI Press, pp. 105–111.

Silver, David (2005). "Cooperative Pathfinding". In: *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005)*, pp. 117–122.

Speck, David (2018). "Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams". MA thesis. University of Freiburg.

Speck, David, Florian Geißer, and Robert Mattmüller (2018). "Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams". In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. Ed. by Mathijs de Weerdt, Sven Koenig, Gabriele Röger, and Matthijs Spaan. AAAI Press.

Srinivasan, Arvind, Timothy Kam, Sharad Malik, and Robert K. Brayton (1990). "Algorithms for Discrete Function Manipulation". In: *Proceedings of the 1990 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1990)*, pp. 92–95.

Sun, Dali, Florian Geißer, and Bernhard Nebel (2016). "Towards effective localization in dynamic environments". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016*, pp. 4517–4523.

Torralba, Álvaro (2015). "Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning". PhD thesis. Universidad Carlos III de Madrid.

Torralba, Álvaro and Florian Pommerening, eds. (2018). *Ninth International Planning Competition (IPC-9): planner abstracts*.

Trevizan, Felipe W., Sylvie Thiébaux, and Patrik Haslum (2017). "Occupation Measure Heuristics for Probabilistic Planning". In: *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*. Ed. by Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith. AAAI Press, pp. 306–315.

Wacker, Erik (2017). "Klassische Planung mit zustandsabhängigen Aktionskosten mit und ohne Kompilierung". MA thesis. University of Freiburg.

Yoon, Sung Wook, Alan Fern, and Robert Givan (2007). "FF-Replan: A Baseline for Probabilistic Planning". In: *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*. Ed. by Mark Boddy, Maria Fox, and Sylvie Thiébaux. AAAI Press, pp. 352–360.

Yoon, Sung Wook, Alan Fern, Robert Givan, and Subbarao Kambhampati (2008). "Probabilistic Planning via Determinization in Hindsight". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*. AAAI Press, pp. 1010–1016.

Younes, Håkan L. S. and Michael L. Littman (2004). *PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects*. Tech. rep. CMU-CS-04-167. Carnegie Mellon University, School of Computer Science.