Delete Relaxations for Planning with State-Dependent Action Costs

Florian Geißer and **Thomas Keller** and **Robert Mattmüller**
University of Freiburg, Germany
{geisserf,tkeller,mattmuel}@informatik.uni-freiburg.de

Abstract

Most work in planning focuses on tasks with state-independent or even uniform action costs. However, supporting state-dependent action costs admits a more compact representation of many tasks. We investigate how to solve such tasks using heuristic search, with a focus on delete-relaxation heuristics. We first define a generalization of the additive heuristic h^{add} to such tasks and then discuss different ways of computing it via compilations to tasks with state-independent action costs and more directly by modifying the relaxed planning graph. We evaluate these approaches theoretically and present an implementation of h^{add} for planning with state-dependent action costs. To our knowledge, this gives rise to the first approach able to handle even the hardest instances of the combinatorial ACADEMIC ADVISING domain from the International Probabilistic Planning Competition (IPPC) 2014.

1 Introduction

Many extensions have been made to the Planning Domain Definition Language (PDDL), not only to allow for more expressivity but also to achieve a modeling of domains which is closer to the real world. Some of these extensions allow domains with state-dependent action costs (SDAC) [Ivankovic *et al.*, 2014], for example unrestricted numeric PDDL [Fox and Long, 2003] or PDDL3 [Gerevini *et al.*, 2009]. E. g., SDAC allow representation of the fuel consumption of a car that depends on its speed and its weight more compactly than with state-independent costs. Ivankovic *et al.* [2014] analyze the power supply restoration domain, where the objective is to minimize the unsupplied load at each plan step and the cost is modeled by a finite sum of conditional costs over the state variables. They simplify the problem by assuming minimal constant action costs and compute an optimal plan for the simplified problem. If the assumed minimal cost is unattainable in the original problem, they split the action into two copies and repeat the first step on the modified problem. Unfortunately, they report that the heuristic is not time-efficient, and as a result blind search solves more problems than search with their heuristic. Another class of problems exists in planning with Markov Decision Processes (MDPs), where costs (or rewards) are associated to pairs of actions and states.

In this work we consider action cost functions, representable by simple enumerations of their values, using function terms, or compactly by decision diagrams. For our purposes so-called *edge-valued multi-valued decision diagrams (EVMDDs)* [Lai *et al.*, 1996; Ciardo and Siminiceanu, 2002] are most convenient, because they allow us to *detect, exhibit and exploit structure* in the cost function. After introducing the formal background we propose a naïve compilation into tasks with state-independent action costs. This compilation is infeasible, since it generates exponentially many actions, but can serve as a baseline for more sophisticated approaches. Based on the structure of the EVMDD we introduce a novel compilation technique, which greatly reduces the number of additional actions. Finally, we avoid compilation altogether by integrating the structure of the EVMDD in the relaxed planning graph (RPG), and show the correctness of this approach. We apply this procedure to the ACADEMIC ADVISING domain [Guerin *et al.*, 2012] and compare the results with the standard heuristic of the PROST planner [Keller and Eyerich, 2012], a state-of-the-art planning system for MDPs. We conclude with some remarks about future work.

2 Preliminaries

In this section, we introduce planning tasks with SDAC, recall the definition of the additive heuristic in classical planning, and extend it to tasks with SDAC.

2.1 Planning with State-Dependent Action Costs

We extend the definition of SAS^+ planning tasks [Bäckström and Nebel, 1995] to take SDAC into account.

Definition 1. *A* planning task with SDAC *is a tuple* $\Pi = (\mathcal{V}, A, s_0, s_\star, (c_a)_{a \in A})$ *consisting of the following components:* $\mathcal{V} = \{v_1, \ldots, v_n\}$ *is a finite set of* state variables*, each with an associated finite domain* $\mathcal{D}_v = \{0, \ldots, |\mathcal{D}_v| - 1\}$*. A* fact *is a pair* (v, d)*, where* $v \in \mathcal{V}$ *and* $d \in \mathcal{D}_v$*, and a* partial variable assignment s *over* $\mathcal{V}$ *is a consistent set of facts. If* s *assigns a value to each* $v \in \mathcal{V}$*,* s *is called a* state*. Let* S *denote the set of states of* Π*. A is a set of* actions*, where an action is a pair* $a = \langle pre, eff \rangle$ *of partial variable assignments, called* preconditions *and* effects*. The state* $s_0 \in S$ *is*

called the initial state*, and the partial state $s_\star$ specifies the* goal *condition. Each action $a \in A$ has an associated* cost function $c_a : S \to \mathbb{N}$ *that assigns the application cost of a to all states where a is applicable, and arbitrary values to all other states.*

For propositional variables, i.e. variables v with $\mathcal{D}_v = \{0, 1\}$, we write v and $\neg v$ for $(v, 1)$ and $(v, 0)$, respectively. For states s, we use function notation $s(v) = d$ and set notation $(v, d) \in s$ interchangeably. Typically, each cost function c_a will be specified as a function term that only depends on a subset of the state variables, which we denote by $supp(a) = \{v^a_1, \ldots, v^a_{k^a}\} \subseteq \mathcal{V}$.[1] It will then be convenient to think of c_a as a function $c_a : \mathcal{D}_1 \times \cdots \times \mathcal{D}_{k^a} \to \mathbb{N}$ where $\mathcal{D}_j$ is the domain of v^a_j, $j = 1, \ldots, k^a$. Let S_a be the set of all valuations of the variables occurring in c_a, i.e., the set of all possible argument tuples of c_a, let $F_a = \bigcup_{\hat{s} \in S_a} \hat{s}$ be the set of all facts (v, d) for $v \in S_a$ and $d \in \mathcal{D}_v$, and let $pvars(a)$ be the set of variables mentioned in the precondition of action a. Throughout the paper, we assume without loss of generality that $supp(a) \cap pvars(a) = \emptyset$. This assumption is warranted since, whenever a variable v occurs in the precondition of a, its value d is fixed in every application of a and the value d can be plugged into the cost function c_a, thus making c_a independent of v.

The semantics of planning tasks are as usual: An action a is applicable in state s iff *pre* $\subseteq s$. Applying action a to s yields the state s' with $s'(v) = \mathit{eff}(v)$ where $\mathit{eff}(v)$ is defined and $s'(v) = s(v)$ otherwise. We write $s[a]$ for s'. A state s is a goal state if $s_\star \subseteq s$. We denote the set of goal states by $S_\star$. Let $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ be a sequence of actions from A. We call π *applicable* in 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 $i = 0, \ldots, n-1$. We call π a *plan* for Π if it is applicable in s_0 and if $s_n \in S_\star$. The *cost* of plan π is the sum of action costs along the induced state sequence, i.e., $cost(\pi) = c_{a_0}(s_0) + \cdots + c_{a_{n-1}}(s_{n-1})$.

Dealing with SDAC in an explicit-state forward search is straightforward, since upon expansion of a state s, determining the relevant action cost values amounts to a simple function evaluation. However, if the search is guided by a domain-independent distance heuristic h, the action costs should also be correctly reflected by the h values assigned to states. This is where complications arise, at least for delete-relaxation heuristics [Bonet *et al.*, 1997; Bonet and Geffner, 1999; 2001] such as the additive heuristic h^{add}.

2.2 Delete-Relaxation Heuristics

A delete-relaxation of a planning task ignores delete effects, which translates to allowing state variables to hold several values simultaneously. Thus, in the relaxed version of an SAS^+ planning task, state variables accumulate their values rather than switching between them.

[1] There can also be variables *within* $supp(a)$ that have no influence on the cost of a. E.g., for $c_a = B + A - A^2$ and $\mathcal{D}_A = \{0, 1\}$, $A \in supp(a)$, although A and A^2 cancel out and c_a is in fact independent of the value of A. We accept this since a more "semantic" definition of the support easily becomes computationally intractable.

Formally, a *relaxed planning task with SDAC* Π^+ consists of the same variables $\mathcal{V}$, the same actions A, and the same goal condition $s_\star$ as the unrelaxed task Π. A *relaxed state* s^+ now assigns to each $v \in \mathcal{V}$ a non-empty subset $s^+(v) \subseteq \mathcal{D}_v$. The set S^+ is the set of all relaxed states, and for $s \in S$, $s^+ \in S^+$ we say that s^+ *subsumes* s iff for all $v \in \mathcal{V}$, $s(v) \in s^+(v)$. An action $a = \langle \mathit{pre}, \mathit{eff} \rangle$ is *relaxed applicable* in s^+ iff $\mathit{pre}(v) \in s^+(v)$. Applying a in s^+ results in $s^{+\prime}$ with $s^{+\prime}(v) = s^+(v) \cup \{\mathit{eff}(v)\}$ if $\mathit{eff}(v)$ is defined and $s^{+\prime}(v) = s^+(v)$ otherwise. The initial state s^+_0 assigns to each variable v the singleton set $\{s_0(v)\}$. Relaxed plans are then defined in the obvious way. What is still missing from this definition is the extension of the action cost function to relaxed states. Here, we define the cost of a in s^+, $c_a(s^+)$, in a natural way as the *minimal* cost $c_a(s)$ for any unrelaxed state s that is subsumed by s^+. The problem with this definition is not that much a conceptual, but rather a computational one. With n state variables, there can be exponentially many unrelaxed states s subsumed by s^+. In this paper we present a representation of cost functions that is typically (although not always) compact and that allows a simple computation of $c_a(s^+)$ and a simple integration into the RPG.

Delete-relaxation heuristics can be generalized to tasks with SDAC: For h^+, generalization is straightforward: $h^+(s)$ is the cost of a cost-minimal relaxed plan starting in s^+. However, the computation of the generalized h^+ heuristic is still NP-hard, as it is already for classical planning [Bylander, 1994]. Thus, we also want to generalize approximations to h^+, such as h^{add} and h^{max}, to tasks with SDAC. In the following, we discuss h^{add} and leave a detailed discussion of h^{max} for future work. For a fact $f = (v, d)$, let $A(f)$ be the set of *achievers* of f, i.e., the set of actions $a = \langle \mathit{pre}, \mathit{eff} \rangle \in A$ with $f \in \mathit{eff}$. Then for classical planning, h^{add} is defined as follows [Bonet *et al.*, 1997; Bonet and Geffner, 1999]:

$$h^{add}(s) = h^{add}_s(s_\star) \tag{1}$$

$$h^{add}_s(s_\mathrm{p}) = \sum_{f \in s_\mathrm{p}} h^{add}_s(f) \quad \text{and} \tag{2}$$

$$h^{add}_s(f) = \begin{cases} 0 & \text{if } f \in s \\ \min\limits_{a \in A(f)} \left[h^{add}_s(\mathit{pre}(a)) + c_a \right] & \text{otherwise,} \end{cases} \tag{3}$$

where s_p stands for a partial state, f for a fact, $\mathit{pre}(a)$ for the precondition of a, and c_a is the *state-independent* cost of a. A natural generalization to SDAC should, in case (3), minimize over all achievers a of f *and over all possible situations where a is applicable*.[2] Thus, cases (1) and (2) remain unchanged, and case (3) is replaced by case (3′) below. Then:

$$C^a_s = \min_{\hat{s} \in S_a} [c_a(\hat{s}) + h^{add}_s(\hat{s})] \quad \text{in the recursive call}$$

$$h^{add}_s(f) = \begin{cases} 0 & \text{if } f \in s \\ \min\limits_{a \in A(f)} \left[h^{add}_s(\mathit{pre}(a)) + C^a_s \right] & \text{otherwise.} \end{cases} \tag{3′}$$

[2] Consider state s with $s(A) = s(B) = 0$, actions $a_1 = \langle \emptyset, B \rangle$ with $c_{a_1} = 2 - 2 \cdot A$ and $a_2 = \langle \emptyset, A \rangle$ with $c_{a_2} = 1$. We can achieve B at cost 2 in one step, or at cost $0 + 1 = 1$ in two steps if $h^{add}_s(A) = 1$ and we take this h^{add} value into account.

The only difference between Equations (3) and (3′) is the replacement of the constant c_a by the term C_s^a. Obviously, if c_a is a constant function, as in classical planning, $C_s^a = c_a(\emptyset) + h_s^{add}(\emptyset) = c_a$, and for classical tasks, our generalized definition of h^{add} becomes the standard definition again. The central new challenge is the efficient computation of C_s^a. Unfortunately, the number of states in S_a is exponential in the number of variables in $supp(a)$, and in general, C_s^a cannot be additively decomposed by the variables in $supp(a)$. However, we can represent c_a and C_s^a with the help of edge-valued decision diagrams such that c_a and C_s^a are efficiently computable in the size of the decision-diagram representation and that the size of the representation itself is, although exponentiality in the size of $supp(a)$ cannot be avoided in general, compact in many "typical", well-behaved cases.

2.3 Edge-Valued Decision Diagrams

Each action cost function $c_a : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathbb{N}$ over variables $\mathcal{V} = \{v_1, \ldots, v_n\}$ with domains $\mathcal{D}_v = \{0, \ldots, |\mathcal{D}_v| - 1\}$ can be encoded as an edge-valued multi-valued decision diagram (EVMDD) [Ciardo and Siminiceanu, 2002]. Formally, we use a definition of EVMDDs similar to the EVBDD definition by Lai et al. [Lai *et al.*, 1996].

Definition 2. *An* EVMDD *over $\mathcal{V}$ is a tuple $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ where $\kappa \in \mathbb{Z}$ is a constant value and $\mathbf{f}$ is a directed acyclic graph consisting of two types of nodes:*

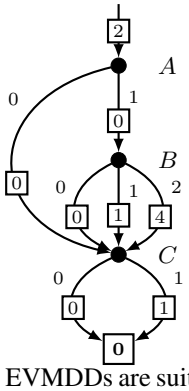
1. *There is a single* terminal node *denoted by* $\mathbf{0}$.
2. *A* nonterminal node $\mathbf{v}$ *is a tuple $(v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ where $v \in \mathcal{V}$ is a variable, $k = |\mathcal{D}_v| - 1$, children $\chi_0, \ldots, \chi_k$ are terminal or nonterminal nodes of $\mathcal{E}$ and $w_1, \ldots, w_k \in \mathbb{Z}$, $w_0 = 0$.*

We refer to the components of $\mathbf{v}$ as $v(\mathbf{v}), \chi_i(\mathbf{v})$ 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 $w_i(\mathbf{v})$. The following definition specifies the arithmetic function denoted by a given EVMDD.

Definition 3. *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 S given by (I) $f(s) = f_{s(v)}(s) + w_{s(v)}$, where $f_{s(v)}$ is the arithmetic function denoted by child $\chi_{s(v)}$. We write $\mathcal{E}(s)$ for $\kappa + f(s)$.*

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 $\mathbf{0}$. A nonterminal node is a tuple $(v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$, where v is the node label, $\chi_0, \ldots, \chi_k$ are the subgraphs rooted at v, and $w_0, \ldots, w_k$ are the weights assigned to the edges of v. Edge labels d are written next to the edges, edge weights w_d in boxes on the edges, $d = 0, \ldots, k$. For fixed variable orders, reduced and ordered EVMDDs are unique [Ciardo and Siminiceanu, 2002]. However, in our application, we do not care about uniqueness, and we can even use different, appropriate variable orderings for different cost functions. Notice that, for appropriate variable orders, the encoding size of an EVMDD $\mathcal{E}_a$ for cost function c_a is polynomial in the number of variables in $supp(a)$ for many types of cost functions: For $\sum_{j=1}^{n} \kappa_j v_j$ with natural coefficients κ_j it is linear, and for general multivariate polynomials over $supp(a)$, it is still only exponential in the size of the largest cluster of variables from $supp(a)$ that are interdependent in c_a.

Example 1. *Consider the action cost function $c_a = AB^2 + C + 2$ with $\mathcal{D}_A = \mathcal{D}_C = \{0, 1\}$, and $\mathcal{D}_B = \{0, 1, 2\}$.*



Depicted is an EVMDD for c_a and the variable ordering A, B, C. To see how the EVMDD encodes the cost function c_a, consider the valuation s with $s(A) = 1$, $s(B) = 2$ and $s(C) = 0$. Traversing the corresponding edges in the EVMDD from top to bottom and adding up the edge weights, we arrive at the resulting value 6, which is exactly $c_a(s)$.

EVMDDs are suitable for the encoding of c_a since the evaluation of an EVMDD is straightforward both in unrelaxed and in relaxed states. Let $\mathcal{E}_a$ be an EVMDD encoding c_a. For an unrelaxed state s, $c_a(s)$ can be read from $\mathcal{E}_a$ as the path cost of the unique EVMDD path corresponding to s, $\mathcal{E}_a(s)$. For a relaxed state s^+, $c_a(s^+)$ can be determined by traversing $\mathcal{E}_a$ in a topological order, only following edges "subsumed" by s^+, and *minimizing* over accumulated costs of multiple incoming arcs of the same decision node:

Definition 4. *Similar to Definition 3, we can define the arithmetic function denoted by $\mathcal{E}$ for* relaxed *state s^+ by replacing Equation (I) with (II) $f(s^+) = \min_{d \in s^+(v)}(f_d(s^+) + w_d)$. We write $\mathcal{E}(s^+)$ for $\kappa + f(s^+)$.*

Theorem 1. *$\mathcal{E}_a(s^+) = c_a(s^+)$ for action cost function c_a, EVMDD $\mathcal{E}_a$ for c_a, and relaxed state s^+ over $supp(a)$.*

Proof sketch. States subsumed by s^+ correspond to paths in $\mathcal{E}_a$ minimized over in the definition of $\mathcal{E}_a(s^+)$. □

This shows that EVMDDs allow an efficient computation of c_a values of unrelaxed and relaxed states. In order to show that EVMDDs can also be used to compute C_s^a, we need to incorporate h_s^{add} values into $\mathcal{E}_a$. To do this correctly, it is necessary that on each path through $\mathcal{E}_a$, each variable in $supp(a)$ is tested, since otherwise the h_s^{add} values of facts for variables skipped on some branch would not be included there. Hence, in the following we assume that $\mathcal{E}_a$ includes branches over all variables on all paths, even if this does not affect the value of $\mathcal{E}_a$. In the EVMDD from Example 1, this means that there is also a test branching over variable B instead of the edge from A immediately to C, where all three B branches have weight 0. We call such an $\mathcal{E}_a$ *quasi-reduced* and note that turning a given reduced EVMDD into this form only leads to a polynomial blowup in the worst case.

3 Action Compilations

A simple way of getting rid of SDAC in a principled manner is by translating Π into an equivalent task where action costs

are state independent, and then to evaluate h on the compiled task. In this section, we discuss two such compilations.

3.1 Basic Action Compilation

In the basic compilation, each action a is transformed into multiple actions $a^{\hat{s}}$, corresponding to the valuations $\hat{s}$ of the variables occurring in c_a, such that $a^{\hat{s}}$ is only applicable in $\hat{s}$. Formally, for every assignment $\hat{s} \in S_a$, an action $a^{\hat{s}} = \langle \mathit{pre}(a) \wedge \bigwedge_{f \in \hat{s}} f, \mathit{eff} \rangle$ with cost $c(a^{\hat{s}}) = c_a(\hat{s})$ is added to the compiled task, where $\mathit{pre}(a)$ is the precondition of a. We call this the basic compilation scheme (BC). It is not hard to see that this leads to an exponential blowup in the number of variables in $\mathit{supp}(a)$, as discussed by Ivankovic *et al.* [2014].

Example 2. *Consider again action a with $c_a = AB^2 + C + 2$ as defined in Example 1. Applying BC on a results in*

$$\begin{aligned}
a^{\neg A, B=0, \neg C} &= \langle \mathit{pre} \wedge \neg A \wedge B = 0 \wedge \neg C, \mathit{eff} \rangle, & \mathit{cost}: 2 \\
a^{\neg A, B=0, C} &= \langle \mathit{pre} \wedge \neg A \wedge B = 0 \wedge C, \mathit{eff} \rangle, & \mathit{cost}: 3 \\
a^{\neg A, B=1, \neg C} &= \langle \mathit{pre} \wedge \neg A \wedge B = 1 \wedge \neg C, \mathit{eff} \rangle, & \mathit{cost}: 2 \\
&\dots \\
a^{A, B=2, \neg C} &= \langle \mathit{pre} \wedge A \wedge B = 2 \wedge \neg C, \mathit{eff} \rangle, & \mathit{cost}: 6 \\
a^{A, B=2, C} &= \langle \mathit{pre} \wedge A \wedge B = 2 \wedge C, \mathit{eff} \rangle, & \mathit{cost}: 7
\end{aligned}$$

On the other hand, the basic compilation scheme produces the desired h^{add} values.

Theorem 2. *Let Π be a task with SDAC and Π' its BC compilation. Let s be a state of Π and h^{add} and $h^{\mathit{add}}_{\mathit{orig}}$ the generalized and original additive heuristics for tasks Π and Π', respectively. Then $h^{\mathit{add}}(s) = h^{\mathit{add}}_{\mathit{orig}}(s)$.*

Proof sketch. Follows from the original and generalized definitions of h^{add} and of the BC compilation, as states $\hat{s}$ correspond uniquely to actions $\hat{a}$ in the basic compilation. □

Instead of further studying BC, we only treat it as a baseline approach and define a more compact EVMDD-based compilation instead.

3.2 EVMDD-Based Action Compilation

To compute the cost of a state in the EVMDD, one has to sum up the costs of the edges on the path from the root node to the terminal node. The key insight is that the 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 has the value corresponding to the edge. Based on this observation we compile the structure of the EVMDD into auxiliary actions. 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. Let $\mathcal{E}_a = \langle \kappa, \mathbf{f} \rangle$ be a quasi-reduced EVMDD for some cost function c_a, $|\mathbf{f}|$ the number of nodes of $\mathcal{E}_a$, and let $\mathit{idx}(\mathbf{v}) \in \{1, \dots, |\mathbf{f}|\}$ be a topological ordering of $\mathcal{E}_a$, i.e., a bijective function numbering the nodes of $\mathcal{E}_a$ such that for every non-terminal node $\mathbf{v}$ and each child $\chi_i(\mathbf{v})$, $\mathit{idx}(\mathbf{v}) < \mathit{idx}(\chi_i(\mathbf{v}))$. If $\mathbf{v}$ is the terminal node $\mathbf{0}$, then $\mathit{idx}(\mathbf{v}) = |\mathbf{f}|$. The domain of aux^a has one value for each node in the EVMDD (plus one start value), i.e., $\mathcal{D}_{\mathit{aux}} = \{0, \dots, |\mathbf{f}|\}$. For each non-terminal node $\mathbf{v} = (v, \chi_0, \dots, \chi_k, w_0, \dots, w_k)$ of $\mathcal{E}_a$ and each $i = 0, \dots, k$, we create an action $a^{v=i, \mathit{idx}(\mathbf{v})} = \langle \mathit{aux}^a = \mathit{idx}(\mathbf{v}) \wedge v = i, \mathit{aux}^a = \mathit{idx}(\chi_i(\mathbf{v})) \rangle$ with $c_{a^{v=i, \mathit{idx}(\mathbf{v})}} = w_i$. Additionally, we need two actions corresponding to the precondition and the effect: A start action $a^{\mathit{pre}} = \langle \mathit{pre} \wedge \mathit{aux}^a = 0, \mathit{aux}^a = 1 \rangle$ with cost $c_{a^{\mathit{pre}}} = \kappa$ and a stop action $a^{\mathit{eff}} = \langle \mathit{aux}^a = |\mathbf{f}|, \mathit{eff} \wedge \mathit{aux}^a = 0 \rangle$ with $c_{a^{\mathit{eff}}} = 0$.[3]

Example 3. *Consider again action a with $c_a = AB^2 + C + 2$ encoded in the quasi-reduced form $\mathcal{E}'_a$ of the EVMDD $\mathcal{E}_a$ from Example 1 with an extra test for B on the $(A, 0)$ branch. Applying the EVMDD compilation to a leads, among others, to the following actions (we write aux for aux^a):*

$$\begin{aligned}
a^{\mathit{pre}} &= \langle \mathit{pre} \wedge \mathit{aux} = 0, \mathit{aux} = 1 \rangle, & \mathit{cost}: 2 \\
a^{A, [1]} &= \langle \mathit{aux} = 1 \wedge A, \mathit{aux} = 3 \rangle, & \mathit{cost}: 0 \\
&\dots \\
a^{B=0, [3]} &= \langle \mathit{aux} = 3 \wedge B = 0, \mathit{aux} = 4 \rangle, & \mathit{cost}: 0 \\
a^{B=1, [3]} &= \langle \mathit{aux} = 3 \wedge B = 1, \mathit{aux} = 4 \rangle, & \mathit{cost}: 1 \\
a^{B=2, [3]} &= \langle \mathit{aux} = 3 \wedge B = 2, \mathit{aux} = 4 \rangle, & \mathit{cost}: 4 \\
&\dots \\
a^{\mathit{eff}} &= \langle \mathit{aux} = 5, \mathit{eff} \wedge \mathit{aux} = 0 \rangle, & \mathit{cost}: 0
\end{aligned}$$

The size of the resulting task depends on the number of edges in the EVMDD. This can lead to a much smaller number of new actions than in the BC compilation. For example, to transform an action with cost function $c_a = \sum_{j=1}^{n} v_j$ for propositional variables v_j, $j = 1, \dots, n$, we only need $2n + 2$ actions, whereas the BC compilation produces 2^n new actions. Furthermore, h^{add} applied on an EVMDD-compiled planning task still results in the desired value:

Theorem 3. *Let Π be a task with SDAC and Π' its EVMDD compilation. Let s be a state of Π and $s' = s \cup \bigcup_{a \in A} \{(\mathit{aux}^a, 0)\}$ be the corresponding state in Π', and let h^{add} and $h^{\mathit{add}}_{\mathit{orig}}$ be the generalized and original additive heuristics. Then $h^{\mathit{add}}(s) = h^{\mathit{add}}_{\mathit{orig}}(s')$.*

Proof sketch. We only need to show that Equations (3) and (3′) are equal for s and s'. By construction, we get one $a^{\mathit{eff}} \in A'(f)$ for each $a \in A(f)$. During computation of $h^{\mathit{add}}_{\mathrm{orig}}(\mathit{pre}(a^{\mathit{eff}}))$ we minimize over all paths from the root to $\mathbf{0}$ in the EVMDD. Since each path corresponds to one $\hat{s} \in S_a$ we get $h^{\mathit{add}}_{\mathrm{orig}}(\mathit{pre}(a^{\mathit{eff}})) = h^{\mathit{add}}_s(\mathit{pre}(a)) + C^a_s$. Then we get $h^{\mathit{add}}_{\mathrm{orig}}(f) = \min_{a^{\mathit{eff}} \in A'(f)} [h^{\mathit{add}}_s(\mathit{pre}(a)) + C^a_s]$. Since a^{eff} corresponds to $a \in A(f)$ we get $h^{\mathit{add}}(s) = h^{\mathit{add}}_{\mathrm{orig}}(s')$. □

4 EVMDD-Based RPG Compilation

In the following, we will show that we can also use EVMDDs to compute C^a_s directly, essentially by compiling the EVMDDs as sub-graphs into the RPG. In principle, computation of C^a_s is easy given a quasi-reduced EVMDD $\mathcal{E}_a$: One can construct from $\mathcal{E}_a$ an EVMDD $\mathcal{G}^a_s$ for $c_a + h^{\mathit{add}}_s$ by retaining the graph structure of $\mathcal{E}_a$ and incorporating h^{add} values into the edge weights, and use $\mathcal{G}^a_s$ for minimization over s^+-viable paths. Such a transformation from $\mathcal{E}_a$ to $\mathcal{G}^a_s$ has one drawback: It assumes that, when generating $\mathcal{G}^a_s$, all necessary $h^{\mathit{add}}_s((v, d))$ values to be hard-coded in $\mathcal{G}^a_s$ are already

[3] In an unrelaxed setting, introduction of an additional semaphore variable can help reducing the branching factor of the search.

known. However, due to the recursive nature of the h_s^{add} computation, this is not the case. This problem can be avoided by adding "input nodes" for all relevant $h_s^{add}((v,d))$ values to the EVMDD. Since we will use the resulting "input-node augmented EVMDD" (INADD) as a portion of an RPG later, we do not treat it as an EVMDD any more, but rather as an *AND/OR DAG*.

Definition 5. *Let $\mathcal{E}_a = \langle \kappa, \mathbf{f} \rangle$ be a quasi-reduced EVMDD for some cost function c_a with support $supp(a)$ and F_a the set of facts over $supp(a)$. Then the* input-node augmented decision diagram (INADD) *for $\mathcal{E}_a$ is the directed acyclic AND/OR graph $\mathcal{I}_a = (N_\wedge, N_\vee, E, W)$ with AND nodes $N_\wedge$, OR nodes $N_\vee$, edges E, and node weight function $W : N_\wedge \to \mathbb{Z}$ with the following components:*

- Input nodes: *For each fact $f \in F_a$, there is an input node $n_f \in N_\vee$.*
- Variable nodes: *For each node $\mathbf{v}$ of $\mathcal{E}_a$, there is a node $n_\mathbf{v} \in N_\vee$. This includes a node $n_\mathbf{0}$ for the terminal node $\mathbf{0}$ of $\mathcal{E}_a$.*
- Fact nodes: *For each non-terminal node $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ of $\mathcal{E}_a$ and each $i = 0, \ldots, k$, there is a node $n_{(\mathbf{v},\chi_i,w_i)} \in N_\wedge$ with weight $W(n_{(\mathbf{v},\chi_i,w_i)}) = w_i$. There is a node $n_\kappa \in N_\wedge$ for the additive constant κ with weight $W(n_\kappa) = \kappa$.*
- AND edges from variable nodes: *For each variable node $n_\mathbf{v} \in N_\vee$ for non-terminal $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ and each $i = 0, \ldots, k$, there is an edge from $n_\mathbf{v}$ to $n_{(\mathbf{v},\chi_i,w_i)}$.*
- AND edges from input nodes: *For each input node n_f for fact $f = (v, d)$ and each fact node $n_{(\mathbf{v},\chi_d,w_d)}$ where $v(\mathbf{v}) = v$, there is an edge from n_f to $n_{(\mathbf{v},\chi_d,w_d)}$.*
- OR edges: *For each fact node $n_{(\mathbf{v},\chi_i,w_i)} \in N_\wedge$ for $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$, there is an edge from $n_{(\mathbf{v},\chi_i,w_i)}$ to n_{χ_i}. There is an edge from n_κ to $n_\mathbf{f}$, where $\mathbf{f}$ denotes the root node of $\mathcal{E}_a$.*

This definition is best illustrated with the running example.

Example 4. *Consider the EVMDD $\mathcal{E}_a$ from Example 1 in its quasi-reduced form $\mathcal{E}'_a$. The INADD $\mathcal{I}_a$ for $\mathcal{E}'_a$ is depicted in Figure 1. OR nodes are shown as ellipses, AND nodes as rectangles, input nodes are labeled with the input facts, arcs from input nodes are thin, arcs corresponding to EVMDD arcs are solid, and node weights are given as annotations next to the corresponding nodes. The gray portion results from the extension of $\mathcal{E}_a$ to $\mathcal{E}'_a$. The red node labels and other red markings will only be of interest in Example 5.*

We can now define the evaluation of an INADD for a given input valuation as follows:

Definition 6. *Let $\mathcal{I}_a = (N_\wedge, N_\vee, E, W)$ be the INADD for some EVMDD $\mathcal{E}_a$ and let ι be a function assigning natural numbers or infinity to input nodes of $\mathcal{I}_a$. Then the node valuation $\hat{\iota}$ of all nodes of $\mathcal{I}_a$ is recursively defined as follows:*

- *For input node $n \in N_\vee$, $\hat{\iota}(n) = \iota(n)$.*
- *For variable node $n \in N_\vee$, $\hat{\iota}(n) = \min_{(n',n)\in E} \hat{\iota}(n')$.*
- *For fact node $n \in N_\wedge$, $\hat{\iota}(n) = \sum_{(n',n)\in E} \hat{\iota}(n') + W(n)$.*

The value *of $\mathcal{I}_a$ for ι, denoted as $\mathcal{I}_a(\iota)$, is then the value $\hat{\iota}(n_\mathbf{0})$ of the INADD node corresponding to the terminal node $\mathbf{0}$ of $\mathcal{E}_a$. Since $\mathcal{I}_a$ is acyclic, this is well-defined.*

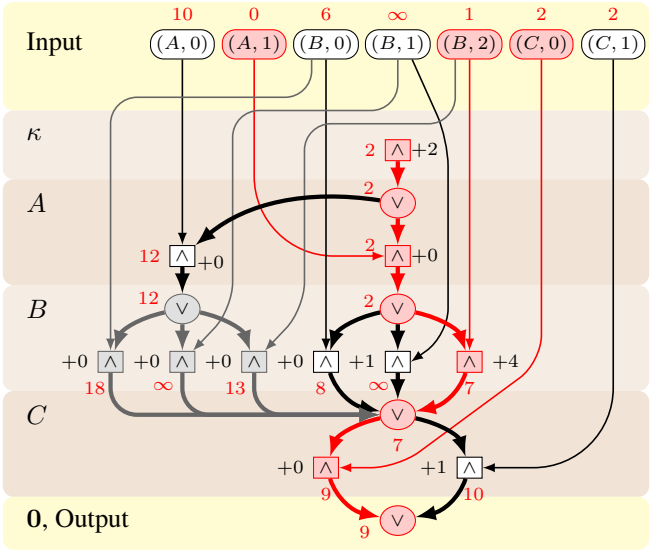


Figure 1: The INADD $\mathcal{I}_a$ for $\mathcal{E}'_a$

Example 5. *Consider the INADD from Example 4. The red input node labels denote $\iota(n)$ for some input valuation ι, and the red interior node labels denote the corresponding valuation $\hat{\iota}$. The highlighted red path of interior nodes is the minimizing path determining $\mathcal{I}_a$, with inputs used on that path also highlighted in red. As we will see formally below, the value $\mathcal{I}_a = 9$ is the same as $C_s^a = \min_{\hat{s}\in S_a}[c_a(\hat{s}) + h_s^{add}(\hat{s})]$ if the ι values are interpreted as h_s^{add} values: The highlighted path corresponds to the minimizing $\hat{s}$, $c_a(\hat{s})$ is the sum of the node weights $2 + 0 + 4 + 0 = 6$ along the path, and $h_s^{add}(\hat{s})$ is the sum of used input node costs, $0 + 1 + 2 = 3$.*

We can now use $\mathcal{I}_a$ to compute C_s^a by initializing the valuation of input nodes with the h^{add} values of the corresponding facts. Efficient computability in the size of $\mathcal{I}_a$ is obvious.

Lemma 1. *Let $\mathcal{I}_a = (N_\wedge, N_\vee, E, W)$ be the INADD for a quasi-reduced EVMDD $\mathcal{E}_a$ encoding the action cost function c_a of action a, and let $v_1 \prec \cdots \prec v_k$ be the variable order of $\mathcal{E}_a$. Let $\mathcal{C}_s^a$ be the function over the variable nodes $n = n_\mathbf{v}$ in $N_\vee$ defined recursively as follows:*

- *Base case for $v(\mathbf{v}) = v_1$ or $k = 0$ and $\mathbf{v} = \mathbf{0}$: $\mathcal{C}_s^a(n) = \min_{(n',n)\in E} W(n')$.*
- *Inductive case for $v(\mathbf{v}) = v_j$, $j > 1$, or $k > 0$ and $\mathbf{v} = \mathbf{0}$: $\mathcal{C}_s^a(n) = \min_{(n',n)\in E}[\mathcal{C}_s^a(n'_\vee) + h_s^{add}(f) + W(n')]$, where $n'_\vee \in N_\vee$ is the unique OR parent of n' and f is the fact represented by the input node connected to n'.*

Then we get the following result:

1. *$\mathcal{C}_s^a(n) = \min_{\hat{s}\in S_n}[\hat{c}_a(\hat{s}) + h_s^{add}(\hat{s})]$, where S_n is the set of partial states corresponding to paths incoming in n and where $\hat{c}_a(\hat{s})$ is c_a applied to $\hat{s} \cup \{(v_i, 0) \mid v_i \in supp(a)$ not occurring in $\hat{s}\}$.*
2. *$\mathcal{C}_s^a(n_\mathbf{0}) = C_s^a$.*

Proof sketch. 1. Induction over recursive definition of $\mathcal{C}_s^a(n)$. In the base case, both sides evaluate to κ. For the inductive case, we can apply the induction hypothesis to

Instance	1	2	3	4	5	6	7	8	9	10
Variables	20	20	30	30	40	40	50	50	60	60
Actions	11	11	16	16	21	21	26	26	31	31
Variables in c_a	4	8	5	8	9	11	9	10	12	12
Size of compilation	168	2 688	496	3 968	10 496	41 984	13 056	26 112	124 928	124 928
Size of $\mathcal{I}_a$	$14+17$	$26+33$	$17+21$	$26+33$	$29+37$	$35+45$	$29+37$	$32+41$	$38+49$	$38+49$
baseline	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00
IDS	**40.71**	**46.24**	**39.88**	202.19	202.07	202.90	201.65	201.67	201.28	201.51
h^{add}	**40.71**	**45.80**	**40.60**	**63.41**	**130.11**	**76.15**	**105.37**	**109.02**	**180.41**	**125.52**

Table 1: Statistics and results of the ACADEMIC ADVISING domain.

$\mathcal{C}^a_s(n'_\vee)$, use the independence of $h^{add}_s(f)$ and $W(n')$ of the choice of $\hat{s} \in S_{n'_\vee}$, construct $\hat{s}_f(n') = \hat{s} \cup \{f\}$, use $\hat{c}_a(\hat{s}_f(n')) = \hat{c}_a(\hat{s}) + W(n')$ and $h^{add}_s(\hat{s}_f(n')) = h^{add}_s(\hat{s}) + h^{add}_s(f)$, and simplify.

2. By part 1 of this lemma, using $S_{n_\mathbf{0}} = S_a$, which holds since $\mathcal{E}_a$ is quasi-reduced. □

Theorem 4. *Let $\mathcal{I}_a$ be the INADD for a quasi-reduced EVMDD $\mathcal{E}_a$ encoding the action cost function c_a of action a. Let $\iota(n_f) = h^{add}_s(f)$ for all facts $f \in F_a$. Then $\mathcal{I}_a(\iota) = C^a_s$.*

Proof sketch. For non-input OR nodes n, we show by induction over the distance from n to n_κ that $\hat{\iota}(n) = \mathcal{C}^a_s(n)$ (with $\mathcal{C}^a_s$ recursively defined as in Lemma 1). Then the claim follows for $n = n_\mathbf{0}$ using the definition of $\mathcal{I}_a(\iota)$ (left-hand side) and part 2 of Lemma 1 (right-hand side). In the base case, again both sides evaluate to κ. In the inductive case, let $n \in N_\vee$ be a node with distance d. Then $\hat{\iota}(n) = \hat{\iota}(n')$ where $n' \in N_\wedge$ is a $\hat{\iota}$-minimizing fact node with $(n, n') \in E$. Now, n' has two parents, one variable node $n'_\vee$ with strictly smaller distance, and one input node n_f. Thus, by definition, $\hat{\iota}(n') = \hat{\iota}(n'_\vee) + \hat{\iota}(n_f) + W(n')$. By induction hypothesis and the definition of $\hat{\iota}$ for input nodes, this is $\mathcal{C}^a_s(n'_\vee) + h^{add}_s(f) + W(n')$. Thus, $\hat{\iota}(n) = \min_{(n',n)\in E}[\mathcal{C}^a_s(n'_\vee) + h^{add}_s(f) + W(n')] = \mathcal{C}^a_s(n)$ by the choice of n' and recursive definition of $\mathcal{C}^a_s$. □

We can now use the construction of $\mathcal{I}_a$ as a portion of the RPG for Π as follows: In each RPG layer, instead of having a single action node for action a, we have an action sub-graph for a that is the "conjunction" of $\mathcal{I}_a$ with the precondition facts of a. The input nodes of $\mathcal{I}_a$ are the corresponding fact nodes from the previous layer, and the "output" node $n_\mathbf{0}$ of $\mathcal{I}_a$ is linked to the effect literals of a on the current layer.

5 Experimental Evaluation

To evaluate h^{add} on planning tasks with SDAC, we have implemented the heuristic in the 2014 version of the PROST planner and compared it against the standard heuristic of PROST on the ACADEMIC ADVISING domain. The reason we chose the probabilistic setting is that planners already have to be able to deal with SDAC, which are part of the last two IPPCs' benchmark sets. ACADEMIC ADVISING is not the only goal-oriented domain, but the others are either very small, based on constant action costs, or require a transformation to positive action costs. Furthermore, it was by far the most challenging domain of the last IPPC, with no participant able to generate a good policy for more than the first four instances. PROST 2014 is based on the UCT* algorithm [Keller and Helmert, 2013], a Trial-Based Heuristic Tree Search algorithm using UCB1 [Auer *et al.*, 2002] action selection, Monte-Carlo simulation of stochastic outcomes, and a partial Bellman backup function to propagate collected information in the search tree. The standard heuristic (IDS) performs a lookahead based on a sequence of iterative deepening searches on the most-likely determinization of the given MDP. If the search uses too much time, it is stopped prematurely. The challenging part for the ACADEMIC ADVISING domain is that the application of good actions only pays off when the goal is reached, which is why it is costly to gain informative goal distance heuristic values. Table 1 shows statistics for different instances of ACADEMIC ADVISING. The first two rows denote the number of variables and the number of actions, while the third row shows the number of variables occurring in the cost function of each action. Note that this number is identical for all actions except for the no-op action. Next we denote the number of actions in the BC compilation, showing clearly that BC is infeasible. Row five denotes the number of nodes plus the number of edges of the resulting INADD[4]. The last three rows show the average results of PROST across 100 runs for each instance, based on different heuristics, with a 1-second timeout for each planning step. Best results (up to insignificant differences) are bold. *baseline* is based on a policy which always performs the no-op action, IDS is the standard heuristic used by PROST as described by Keller and Eyerich [2012], and h^{add} is the additive heuristic using the EVMDD-based RPG compilation of Section 4. Both IDS and h^{add} are applied to the most-likely determinization. For small problems, IDS performs reasonably well, but suffers from uninformative heuristic values as soon as the problem size increases, which leads to results worse than those of the baseline policy, whereas h^{add} yields good results, even for the most complex instances.

We briefly want to mention some implementation details. Since IPPC domains are modeled in RDDL [Sanner, 2010], we transformed grounded actions of the most-likely determinization into precondition-effect pairs. This transformation leads to more actions (roughly 5 times more), but the generated actions are only used for the computation of the

[4]Note that (i) input nodes are not counted, as they are already part of the RPG anyway, and (ii) the used EVMDDs are not quasi-reduced. We internally transform them from reduced to quasi-reduced on the fly.

heuristic values. EVMDDs are created by the open source MDD library MEDDLY [Badar and Miner, 2011]. Finally we want to mention that even-numbered instances allow concurrent actions. In this case, PROST generates additional actions for all possible combination of actions. During the computation of h^{add}, however, we only reason about the original actions, which leads to a small underestimation of the true h^{add} values but largely reduces the size of the RPG.

6 Conclusion and Future Work

We have shown how action cost functions in planning with SDAC can be compactly encoded using EVMDDs and how a compact compilation to tasks with state-independent action costs can be derived from such an encoding. Moreover, we have introduced a natural extension of the definition of the h^{add} heuristic to tasks with SDAC, which can be efficiently computed, again using the EVMDD structures of the action cost functions. Finally, we have presented an implementation of the generalized h^{add} heuristic in the context of probabilistic planning, where SDAC are common. This heuristic gives rise to the first approach able to handle even the hardest instances of the combinatorial ACADEMIC ADVISING domain. For future work, we plan to extend the empirical evaluation to a broader range of benchmark problems, from probabilistic planning as well as from classical planning with SDAC. Furthermore, we will study the usefulness of the EVMDD encoding for other relaxation heuristics such as h^{max} (which is not as straightforward as it may seem) as well as to other types of heuristics, such as abstractions, and integrate them in the framework of the PROST planner. Finally, we will investigate how to automatically derive suitable EVMDD variable orderings for given cost functions.

Acknowledgements. This work was partly supported by the DFG as part of the SFB/TR 14 AVACS and by BMBF grant 02PJ2667 as part of the KARIS PRO project. We thank the anonymous reviewers for their insightful comments.

References

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Journal of Machine Learning Research*, 47:235–256, May 2002.

[Bäckström and Nebel, 1995] Christer Bäckström and Bernhard Nebel. Complexity Results for SAS+ Planning. *Computational Intelligence*, 11:625–656, 1995.

[Badar and Miner, 2011] Junaid Badar and Andrew Miner. MEDDLY: Multi-terminal and Edge-valued Decision Diagram LibrarY. `http://meddly.sourceforge.net/`, 2011. Accessed: 2015-03-27.

[Bonet and Geffner, 1999] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *Proceedings of the 5th European Conference on Planning (ECP 1999)*, pages 359–371, 1999.

[Bonet and Geffner, 2001] Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence (AIJ)*, 129:5–33, 2001.

[Bonet *et al.*, 1997] Blai Bonet, Gábor Loerincs, and Hector Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997)*, pages 714–719, 1997.

[Bylander, 1994] Tom Bylander. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence (AIJ)*, 69:165–204, 1994.

[Ciardo and Siminiceanu, 2002] Gianfranco Ciardo and Radu Siminiceanu. Using edge-valued decision diagrams for symbolic generation of shortest paths. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2002)*, pages 256–273, 2002.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124, 2003.

[Gerevini *et al.*, 2009] Alfonso E. Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence Journal (AIJ)*, 173(5–6):619–668, 2009.

[Guerin *et al.*, 2012] Joshua T. Guerin, Josiah P. Hanna, Libby Ferland, Nicholas Mattei, and Judy Goldsmith. The academic advising planning domain. In *Proceedings of the 3rd Workshop on the International Planning Competition at ICAPS*, pages 1–5, 2012.

[Ivankovic *et al.*, 2014] Franc Ivankovic, Patrik Haslum, Sylvie Thiébaux, Vikas Shivashankar, and Dana S. Nau. Optimal planning with global numerical state constraints. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 2014.

[Keller and Eyerich, 2012] Thomas Keller and Patrick Eyerich. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 119–127. AAAI Press, June 2012.

[Keller and Helmert, 2013] Thomas Keller and Malte Helmert. Trial-based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, pages 135–143, 2013.

[Lai *et al.*, 1996] Yung-Te Lai, Massoud Pedram, and Sarma B. K. Vrudhula. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers*, 45(2):247–255, 1996.

[Sanner, 2010] Scott Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description. 2010.