Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Abstractions for Planning with State-Dependent Action Costs

Florian Geißer
University of Freiburg, Germany
geisserf@informatik.uni-freiburg.de

Thomas Keller
University of Basel, Switzerland
tho.keller@unibas.ch

Robert Mattmüller
University of Freiburg, Germany
mattmuel@informatik.uni-freiburg.de

Abstract

Extending the classical planning formalism with state-dependent action costs (SDAC) allows an up to exponentially more compact task encoding. Recent work proposed to use edge-valued multi-valued decision diagrams (EVMDDs) to represent cost functions, which allows to automatically detect and exhibit structure in cost functions and to make heuristic estimators accurately reflect SDAC. However, so far only the inadmissible additive heuristic h^{add} has been considered in this context. In this paper, we define informative *admissible* abstraction heuristics which enable *optimal* planning with SDAC. We discuss how *abstract* cost values can be extracted from EVMDDs that represent *concrete* cost functions without adjusting them to the selected abstraction. Our theoretical analysis shows that this is efficiently possible for abstractions that are Cartesian or coarser. We adapt the counterexample-guided abstraction refinement approach to derive such abstractions. An empirical evaluation of the resulting heuristic shows that highly accurate values can be computed quickly.

Introduction

Planning with state-dependent action costs (SDAC) not only leads to a more elegant encoding of planning tasks, but also to an up to exponentially more compact and computationally manageable representation than what one would get by multiplying out all possible valuations of variables on which an action cost function depends. Breaking down higher-level actions with SDAC into low-level micro actions with constant costs is not always possible in a straightforward way, and if done naïvely, it only moves the exponential blow-up from the representation to the search space. Nevertheless, although PDDL supports SDAC (Fox and Long 2003), we are not aware of any planner able to handle action costs that differ between states. One reason is that, while supporting SDAC is trivial in plain forward search, correctly reflecting them in a goal-distance heuristic is not.

However, there has been some recent work on the topic. Ivankovic et al. (2014) circumvent the problem of dealing with SDAC in the heuristic computation by assuming that all actions have minimal cost over all states. They then iteratively compute a relaxed plan, check if the costs coincide, if necessary refine the corresponding action's cost function by splitting the action in two parts, and repeat in this manner until no violation is left. Geißer, Keller, and Mattmüller (2015) propose a variant of the additive heuristic h^{add} (Bonet, Loerincs, and Geffner 1997) that is able to deal with SDAC. Their approach differs from the method of Ivankovic et al. in that they directly deal with SDAC in the heuristic computation, which is achieved by representing action cost functions as edge-valued multi-valued decision diagrams (EVMDDs) (Lai, Pedram, and Vrudhula 1996; Ciardo and Siminiceanu 2002). EVMDDs allow to *detect*, *exhibit* and *exploit* additive structure present in action cost functions. An important property of EVMDDs is that they are *edge-valued*, i.e., the function values they represent are additively distributed along the paths corresponding to the input. As their edges correspond to facts (variable-value pairs), this means that they *localize* and attribute partial costs to facts responsible for them, in general conditionally on other facts branched over earlier in the EVMDD. In this paper, we build on the idea of encoding cost functions with EVMDDs, and use them to compute abstraction heuristics. Unlike h^{add}, our heuristic is admissible and can hence be used for *optimal* planning with SDAC.

We show that, as long as the abstraction under consideration is *Cartesian* (Ball, Podelski, and Rajamani 2001), the attribution of partial costs to facts can be exploited when computing abstract action cost values. This is essentially done by distinguishing facts consistent with the current abstract state from those inconsistent with it during the evaluation of the EVMDD. The same technique also works for all abstractions that are coarser than Cartesian, including projection or pattern-database abstractions (Culberson and Schaeffer 1998; Edelkamp 2001) and domain abstractions (Hernádvölgyi and Holte 2000). Importantly, our approach allows the computation of abstract costs without adaptation of the concrete cost function to the abstraction.

Our contribution is the following: We define abstract planning tasks with SDAC and discuss the specifics of 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, and discuss how abstractions can be used to find an optimal plan for the concrete problem. We generalize the counterexample-guided abstraction refinement (CEGAR) approach for classical planning (Seipp and Helmert 2013) to planning with SDAC and show

empirically that our implementation of the heuristic outperforms other approaches in terms of heuristic accuracy.

Background

Planning with State-Dependent Action Costs

We consider planning tasks with SDAC, and base our work on the formalism of Geißer, Keller, and Mattmüller (2015).

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:* $\mathcal{V} = \{v_1, \dots, v_n\}$ *is a finite set of* state variables*, each with an associated finite domain* $\mathcal{D}_v = \{0, \dots, |\mathcal{D}_v| - 1\}$*. 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 states s, we use function notation $s(v) = d$ and set notation $(v, d) \in s$ interchangeably. Each cost function c_a can be thought of as a function $c_a : \mathcal{D}_1 \times \cdots \times \mathcal{D}_{k^a} \to \mathbb{N}$, where $v_1, \dots, v_{k^a}$ are the variables c_a depends upon, which we also call the *support* $supp(a)$ of c_a, and $\mathcal{D}_j$, $j = 1, \dots, k^a$, are their domains. 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 $supp(a) \cap pvars(a) = \emptyset$. 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 $s'(v) = eff(v)$ where $eff(v)$ is defined, and $s'(v) = s(v)$ otherwise. We write $s[a]$ for s'. A state s is a goal state iff $s_\star \subseteq s$. We denote the set of goal states by $S_\star$. Let $\pi = \langle a_0, \dots, a_{n-1}\rangle$ be a sequence of actions from A. We call π *applicable* in s_0 if there exist states $s_1, \dots, s_n$ such that a_i is applicable in s_i and $s_{i+1} = s_i[a_i]$ for all $i = 0, \dots, n-1$. 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., $cost(\pi) = \sum_{i=0}^{n-1} c_{a_i}(s_i)$.

Edge-Valued Decision Diagrams

Each action cost function $c_a : \mathcal{D}_1 \times \cdots \times \mathcal{D}_n \to \mathbb{N}$ over variables $\mathcal{V} = \{v_1, \dots, v_n\}$ with domains $\mathcal{D}_v = \{0, \dots, |\mathcal{D}_v| - 1\}$ can be encoded as an EVMDD.

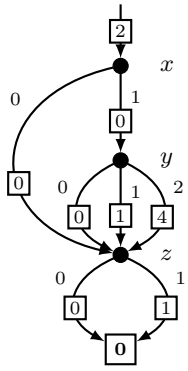
Definition 2. *An* EVMDD *over* $\mathcal{V}$ *is a tuple* $\mathcal{E} = \langle \kappa, \mathbf{f}\rangle$*, where* $\kappa \in \mathbb{Z}$ *is a constant and* $\mathbf{f}$ *is a directed acyclic graph consisting of two types of nodes: (i) there is a single* terminal node *denoted by* $\mathbf{0}$*. (ii) A* nonterminal node $\mathbf{v}$ *is a tuple* $(v, \chi_0, \dots, \chi_k, w_0, \dots, w_k)$ *where* $v \in \mathcal{V}$ *is a variable,* $k = |\mathcal{D}_v| - 1$*, children* $\chi_0, \dots, \chi_k$ *are terminal or nonterminal nodes of* $\mathcal{E}$*, and* $w_0, \dots, w_k \in \mathbb{Z}$ *s.t.* $\min_{i=0,\dots,k} w_i = 0$ *are the weights assigned to the edges to the children.*

By $\mathbf{f}$ we also refer to the root node of $\mathcal{E}$. 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 child χ_i is w_i. 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* $(v, \chi_0, \dots, \chi_k, w_0, \dots, w_k)$ *denotes the arithmetic function over* 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* $\kappa + f(s)$*.*

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. For fixed variable orders, reduced and ordered EVMDDs are unique (Ciardo and Siminiceanu 2002). For suitable variable orders, the encoding size of an EVMDD $\mathcal{E}_a$ for cost function c_a is often polynomial in the number of variables in $supp(a)$.

Example 1. *(From Geißer, Keller, and Mattmüller, 2015.) Consider the action cost function* $c_a = xy^2 + z + 2$ *with* $\mathcal{D}_x = \mathcal{D}_z = \{0, 1\}$ *and* $\mathcal{D}_y = \{0, 1, 2\}$*. The figure on the left depicts an EVMDD for* c_a *and variable order* x, y, z*. To see how the EVMDD encodes the cost function* c_a*, consider the state* s *with* $s(x) = 1$*,* $s(y) = 2$ *and* $s(z) = 0$*. 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* $c_a(s)$*.*



Let $\mathcal{E}_a$ be an EVMDD encoding c_a. For a state s, $c_a(s)$ can be read from $\mathcal{E}_a$ as the path cost of the unique EVMDD path corresponding to s, $\mathcal{E}_a(s)$.

Abstractions for SDAC Tasks

A planning task with SDAC Π induces a transition system $\mathcal{T} = (S, L, T, s_0, S_\star)$ with state space S, transition labels L, transition relation T, initial state s_0 and goal states $S_\star$ in the usual way, with one additional feature: besides source state s, target state t and transition label a, transitions in T also carry a weight $n \in \mathbb{N}$, which is determined by s and a as $n = c_a(s)$. Moreover, given a transition system $\mathcal{T}$, an abstract state space S^α, and a surjective abstraction mapping $\alpha : S \to S^\alpha$, we call $\mathcal{T}^\alpha = (S^\alpha, L, T^\alpha, s_0^\alpha, S_\star^\alpha)$ the abstract transition system induced by $\mathcal{T}$ and α. It is obtained from $\mathcal{T}$ by collapsing concrete states mapped to the same abstract state into one, preserving initial and goal states, and including an abstract transition in T^α iff there is a concrete transition inducing it. We define the weight of an abstract transition between abstract states $\hat{s}$ and $\hat{t}$ with transition label a to be the *minimal* weight of any concrete transition starting in any state s with $\alpha(s) = \hat{s}$ labeled with action a. Notice

that, according to this definition, the cost of an abstract transition only depends on the action label and the source state, not on the target state. This is in accordance with action cost functions only depending on the source, but not the target state, and will be necessary below (cf. Definition 5) when we define an abstract cost function independently of an abstract transition system, and later when we compute abstract costs efficiently using EVMDDs that only have access to the source state, not the target state. It is, however, easily possible to generalize our formalism to cost functions that depend on source and target state, for instance to encode general reward functions of Markov Decision Processes.

In the following, for convenience, we identify abstract states $\hat{s}$ with the sets of concrete states $\{s \in S \mid \alpha(s) = \hat{s}\}$ that are abstracted to them, and use notation like $\alpha(s) = \hat{s}$ and $s \in \hat{s}$ interchangeably. From now on, we focus our attention on so-called *Cartesian abstractions* that, unlike, e.g., projection abstractions, may keep *some*, but not necessarily *all*, information about a variable by partitioning its domain into values consistent with abstract state $\hat{s}$ and values inconsistent with $\hat{s}$. The definition follows Seipp and Helmert (2013).

Definition 4. *A set of states $\hat{s}$ is called* Cartesian *if it is of the form $D_1 \times \cdots \times D_n$, where $D_i \subseteq \mathcal{D}_i$ for all $i = 1, \ldots, n$. An abstraction (mapping) is called* Cartesian *if all its abstract states are Cartesian sets. We call domain values $d \in D_i$* consistent *with $\hat{s}$, and also refer to D_i as $\hat{s}(v_i)$. We assume that $D_i \neq \emptyset$ for all $i = 1, \ldots, n$.*

We will see examples for Cartesian and non-Cartesian abstractions in the following sections. Given a concrete transition system $\mathcal{T}$ (encoded compactly as a planning task Π) and abstraction mapping α, computing the abstract state space, label set and initial state is trivial. For the Cartesian abstractions we are interested in, determining the set of abstract goal states is also easy, as is checking whether action preconditions are satisfied. Since computing abstract state successors is also cheap in a Cartesian setting, we can efficiently determine the set of abstract transitions, except for the correct abstract transition weights. Their efficient computation is the main challenge left, because their weights may not only depend on information that is kept in the abstraction, but also on information that has been abstracted away.

Abstract Cost Computation

Syntactic Cost Minimization

To address the challenge of efficiently computing abstract transition weights, we first define abstract cost functions that syntactically minimize over corresponding concrete costs.

Definition 5. *Let $\Pi = (\mathcal{V}, A, s_0, s_\star, (c_a)_{a \in A})$ be a planning task with SDAC with state space S and $\alpha : S \to S^\alpha$ a surjective abstraction mapping. For each $a \in A$, we define the abstract cost function $c_a^\alpha : S^\alpha \to \mathbb{N}$ as $c_a^\alpha(\hat{s}) = \min_{s \in S : \alpha(s) = \hat{s}} c_a(s)$ for abstract states $\hat{s} \in S^\alpha$.*

It is easy to show that defining c_a^α as above means the same as requiring that abstract transition weights be minimal among all weights of transitions from states abstracted to $\hat{s}$ with the same action label a. In particular, this means that, when we want to compute abstract transition weights, we do *not* have to minimize over all (exponentially many) concrete transitions responsible for that weight, but that it is sufficient to evaluate the abstract cost function $c_a^\alpha(\hat{s})$. Notice that in Definition 5, we do not need to restrict the minimization to states s where action a is applicable, since we assume that $c_a(s)$ is independent of all variables occurring in the precondition of a. Given a plan π, we can now also define the abstract cost of π with α as the sum of the action costs along the induced abstract state sequence, i.e. $cost^\alpha(\pi) = \sum_{i=0}^{n-1} c_{a_i}^\alpha(\hat{s}_i)$

Global EVMDD-Based Cost Minimization

We discuss how to efficiently determine $c_a^\alpha(\hat{s})$ next. First, we give an example that shows that in principle, we need to minimize over exponentially many concrete states in the definition of $c_a^\alpha(\hat{s})$. Then, we show why for Cartesian abstractions, this is unproblematic.

Example 2. *Consider a planning task Π with propositional variables $v_0, \ldots, v_{n-1}, x$, initial state s_0 with all variables zero, goal condition $x = 1$, unit-cost actions for toggling any of the variables v_i (separately), and an action $a^\star$ without precondition that sets x to 1 at cost $c_{a^\star} = 2^n - \sum_{i=0}^{n-1} 2^i v_i$. Consider further the projection abstraction to the variable x that maps all states s with $s(x) = 0$ to the abstract state $\neg\times$, and all states s with $s(x) = 1$ to the abstract state $\times$. Then, apart from irrelevant self-loops induced by the toggling actions, the abstract state space has one nontrivial transition from $\neg\times$ to $\times$ labeled with action $a^\star$. The question is which cost to assign to it. Applying $a^\star$ in the concrete initial state costs 2^n, but with each v_i toggled to 1, $a^\star$ becomes cheaper, with a remaining minimum cost of 1 if all v_i are toggled to 1. There are exponentially many concrete states (hidden in just one abstract state $\neg\times$) and exponentially many different cost values to consider when determining the cost of a single abstract transition.*

As a starting point for our study of the interaction between abstract states and EVMDDs, similar to Definition 3, we define the arithmetic function over abstract states encoded by an EVMDD. Recall that concrete states correspond to paths in EVMDDs. Since we defined abstract action costs by minimizing over concrete states (cf. Definition 5), in order to mirror this definition on the level of EVMDDs, we define the value that an EVMDD assigns to an abstract state $\hat{s}$ as the minimum value over all paths corresponding to $\hat{s}$.

Definition 6. *Let $\mathcal{E}$ be an EVMDD and α an abstraction mapping. The arithmetic function in terms of* global minimization *over abstract states wrt. α is given by $\mathcal{E}^{\alpha,\mathrm{G}}(\hat{s}) = \min_{s \in S : \alpha(s) = \hat{s}} \mathcal{E}(s)$ which amounts to minimizing over all paths in $\mathcal{E}$ that correspond to a concrete state $s \in S$ with $\alpha(s) = \hat{s}$.*

It immediately follows from Definitions 5 and 6 that EVMDDs for concrete cost functions also correctly encode abstract cost values, since the minimizations over concrete states s abstracted to $\hat{s}$ in both definitions are in an exact correspondence.

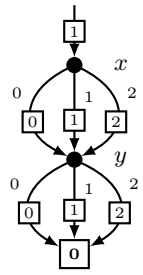
Proposition 1. *Let c_a be the cost function of some action a, let $\mathcal{E}_a$ be an EVMDD encoding c_a, and let $\hat{s}$ be an abstract state wrt. some abstraction mapping α. Then $\mathcal{E}_a^{\alpha,\mathrm{G}}(\hat{s}) = c_a^\alpha(\hat{s})$.* □

In the next section, we will see that this *global* minimization over *paths* can be replaced by a *local* minimization over *edges* if the abstraction is Cartesian. Therefore, computing $\mathcal{E}_a^{\alpha,\mathrm{G}}(\hat{s})$ is cheap if $\hat{s}$ is a Cartesian state, whereas it remains potentially expensive, otherwise.

Local EVMDD-Based Cost Minimization

If the abstraction we face is *not Cartesian*, essentially, the problem that may arise is this: two or more variables can be *independent* in the cost function, giving rise to a compact EVMDD, but *dependent* in the abstraction; then, considering the independent parts of the EVMDD separately is not enough to compute $f(\hat{s})$. Rather, paths have to be investigated in the EVMDD globally. The following example illustrates the issue.

Example 3. *Consider an action a with cost function $c_a = x + y + 1$ for two variables x and y with $\mathcal{D}_x = \mathcal{D}_y = \{0,1,2\}$. The EVMDD for c_a and variable order x, y is depicted to the left. Consider further a (non-Cartesian) abstraction mapping α that maps all states s with $s(x) = s(y)$ to the abstract state $\mathsf{x} = \mathsf{y}$ and all states s with $s(x) \neq s(y)$ to the abstract state $\mathsf{x} \neq \mathsf{y}$. We write 00 for the concrete state s with $s(x) = s(y) = 0$, and correspondingly for all other concrete states. Since $c_a(00) = 1$, $c_a(11) = 3$, and $c_a(22) = 5$, we have $c_a^\alpha(\mathsf{x} = \mathsf{y}) = \min\{1,3,5\} = 1$. In the same vein, we can compute the abstract cost for $x \neq y$ as $c_a^\alpha(\mathsf{x} \neq \mathsf{y}) = 2$. Clearly, all EVMDD edges are consistent with $\mathsf{x} \neq \mathsf{y}$ in the sense that there exists a concrete state s abstracted to $\mathsf{x} \neq \mathsf{y}$ that satisfies the edge constraint (which requires s to assign to the tested variable the value corresponding to the edge). When evaluating the cost in $\mathsf{x} \neq \mathsf{y}$, there is nothing that would make us prefer the $x = 0$, $x = 1$, or $x = 2$ edge over any of the other two edges. The same holds for the $y = 0$, $y = 1$, and $y = 2$ edges, whose consistency with $\mathsf{x} \neq \mathsf{y}$ is* conditional *on which of the three x-edges we traversed. Therefore, we need to consider all paths in the EVMDD that are consistent with $\mathsf{x} \neq \mathsf{y}$. Here, there are six such paths, and in general, there are exponentially many of them in the size of the EVMDD. Independently minimizing over all incoming edges that are consistent with $\mathsf{x} \neq \mathsf{y}$ at all decision points (and the terminal node) would lead to a cost value that* underestimates *the true value $c_a^\alpha(\mathsf{x} \neq \mathsf{y}) = 2$, namely to* 1.



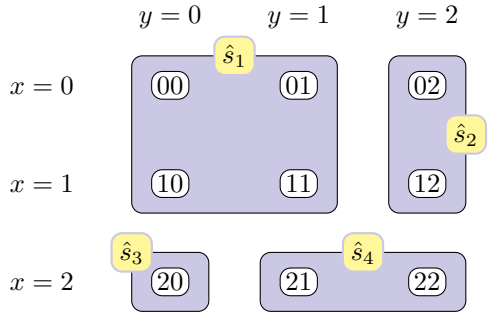
As long as we only care about admissibility of the resulting abstraction heuristic, underestimating costs by independent minimization is sound, but we lose informativeness.

Let us now assume that α is a Cartesian abstraction mapping and that $\hat{s}$ is an abstract state wrt. α. By definition of Cartesian abstractions, for each variable v, an abstract state $\hat{s}$ is consistent with a subset of values $\hat{s}(v)$. In terms of EVMDDs, this means that at a branching point for variable v, state $\hat{s}$ "enables" a subset of the outgoing arcs, exactly those corresponding to values in $\hat{s}(v)$. Therefore, an EVMDD evaluation of $\hat{s}$ that does a single top-sort traversal of $\mathcal{E}$, locally minimizing over all enabled arcs in all decision nodes, correctly computes $\mathcal{E}^{\alpha,\mathrm{G}}(\hat{s})$.

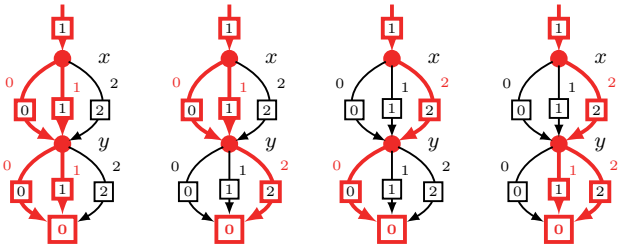
Example 4. *Consider again action a from Example 3, and the abstraction mapping α such that*

$$\alpha(s) = \begin{cases} \hat{s}_1 & \text{if } s(x) \in \{0,1\} \text{ and } s(y) \in \{0,1\}, \\ \hat{s}_2 & \text{if } s(x) \in \{0,1\} \text{ and } s(y) = 2, \\ \hat{s}_3 & \text{if } s(x) = 2 \text{ and } s(y) = 0, \text{ and} \\ \hat{s}_4 & \text{if } s(x) = 2 \text{ and } s(y) \in \{1,2\}. \end{cases}$$

Clearly, this abstraction is Cartesian, and it can be illustrated by the following figure.



Notice that the domain values in the respective sets D_i need not be contiguous. They only happen to be in this example. The figure below shows the EVMDD for c_a (for variable order x, y) with edges consistent with $\hat{s}_1$, $\hat{s}_2$, $\hat{s}_3$, and $\hat{s}_4$, respectively, highlighted in red/bold, from left to right. We can now evaluate this EVMDD by locally minimizing over highlighted incoming arcs.



Close inspection of the four copies of the EVMDD reveals that each of the nine possible paths is highlighted in exactly one of them. Moreover, local minimization leads to values of 1, 3, 3, *and* 4, *respectively (from left to right). Again, these are exactly the abstract cost values $c_a^\alpha(\hat{s}_i)$, $i = 1, \ldots, 4$, as desired.*

Notice that the abstraction from Example 2 is also Cartesian and allows a local minimization as described above, leading to the correct minimum $c_a^\alpha(\neg\mathsf{x}) = 1$.

We now turn to formally stating the previous observations. First, we define the arithmetic function for an abstract state in terms of local minimization.

Definition 7. *Let α be a Cartesian abstraction mapping and $\mathcal{E}$ an EVMDD as in Definition 3. The arithmetic function in*

terms of local minimization *over* abstract *states wrt. α is given by $f^\alpha(\hat{s}) = \min_{d \in \hat{s}(v)}(f_d^\alpha(\hat{s}) + w_d)$, where f_d^α is the arithmetic function denoted by child χ_d. We write $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$ for $\kappa + f^\alpha(\hat{s})$.*

For Cartesian states, global minimization and local minimization indeed return the same result.

Proposition 2. *Let $\mathcal{E}$ be an EVMDD over a set of state variables $\mathcal{V}$, and let $\hat{s}$ be a Cartesian abstract state over a subset of $\mathcal{V}$. Then $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s}) = \mathcal{E}^{\alpha,\mathrm{G}}(\hat{s})$.*

Proof. Both the definition of $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$ and that of $\mathcal{E}^{\alpha,\mathrm{G}}(\hat{s})$ minimize EVMDD path costs over all paths corresponding to concrete valuations s with $\alpha(s) = \hat{s}$. Clearly, all paths minimized over in $\mathcal{E}^{\alpha,\mathrm{G}}(\hat{s})$ are also minimized over in $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$. The converse follows since $\hat{s}$ is Cartesian. □

Notice that this result exactly mirrors our previous result for delete relaxations (Geißer, Keller, and Mattmüller 2015, Theorem 1), since relaxed states are by definition Cartesian sets. Furthermore, we can efficiently compute $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$.

Proposition 3. *Computation of $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$ needs time linear in the size of $\mathcal{E}$.* □

Proof. $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$ can be computed using a single top-sort traversal of $\mathcal{E}$, therefore the cost of computing $\mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$ is linear in the size of $\mathcal{E}$. □

Let us quickly summarize these results and their implications. We can efficiently perform local minimization. For Cartesian states, it does not matter if we perform local or global minimization. Global minimization encodes abstract cost values. We can therefore efficiently compute abstract cost values, which was the last piece required to efficiently compute the abstract transition system. More formally, we get the following corollary from Propositions 1 and 2.

Corollary 1. *Let Π be a planning task with SDAC, let a be an action from Π with cost function c_a, let $\mathcal{E}_a$ be an EVMDD representing c_a, let α be a Cartesian abstraction mapping for Π, and let $\hat{s}$ be an abstract state for mapping α. Then $c_a^\alpha(\hat{s}) = \mathcal{E}^{\alpha,\mathrm{L}}(\hat{s})$.* □

We want to point out that we can, in principle, make sure that each abstraction is Cartesian by merging fluents (van den Briel, Kambhampati, and Vossen 2007). Consider Example 3. After merging x and y, the abstraction is Cartesian. If an abstraction is "k-almost Cartesian", meaning that we need to merge groups of at most k successive fluents in the EVMDD variable order, we also have "k-almost local" minimizations in the unmerged problem. This gives us a similar tractability result as for Cartesian abstractions, now for fixed parameter k.

Abstract Plans

Let us now turn our attention to the relation between a planning task with SDAC and its abstraction. On the one hand, we are interested in the use of an abstraction to compute a *heuristic* estimate to guide search in the concrete task. On the other hand, we want to directly compute optimal plans for the concrete task with the help of the abstraction.

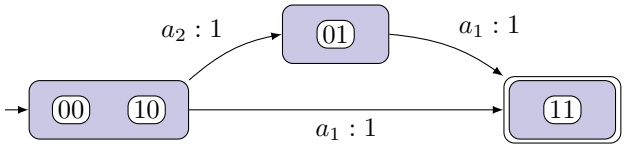
Abstraction heuristics are based on the idea of considering an abstract version of the state space, computing exact goal distances there, and using those goal distances as heuristic estimates in the original problem. In classical planning, abstraction heuristics based on single abstractions are, by construction, guaranteed to be admissible. It is easy to see that this property also holds in the presence of SDAC, as still all concrete paths from $\mathcal{T}$ are preserved as abstract paths in $\mathcal{T}^\alpha$, and concrete plans remain plans in the abstraction. Moreover, since transition costs never increase in the abstraction, path and plan costs never increase, either. In other words, abstraction heuristics based on our abstractions are admissible estimates for the concrete planning task with SDAC.

Proposition 4. *Let $\mathcal{T}$ and $\mathcal{T}^\alpha$ be a transition system and its abstraction, and let π be an optimal abstract plan from an abstract state $\hat{s} \in S^\alpha$. Then $\mathrm{cost}^\alpha(\pi)$ is an admissible heuristic estimate for all $s \in \hat{s}$.* □

We want to remark that all goal distances can be computed with a single sweep over the abstract state space. The second property we are interested in concerns the question when an optimal abstract plan is also an optimal concrete plan. Given an abstract plan $\pi = a_0, \ldots, a_{n-1}$ with induced state sequence $\hat{s}_0, \ldots, \hat{s}_n$ for planning task Π and abstraction mapping α, we call π *concretizable* if π is a plan for Π with induced state sequence $s_0, \ldots, s_n$ such that $\alpha(s_i) = \hat{s}_i$ for $i = 0, \ldots, n$. In the absence of state-dependent action costs, an optimal abstract plan that is concretizable to a plan in the concrete task is also optimal in the concrete setting. If an abstraction is created to provide heuristic values and it turns out that an optimal abstract plan is concretizable, then this solves the concrete task without performing any search in the concrete transition system. Seipp and Helmert (2013), for instance, exploit this property in their empirical evaluation, where they report the number of instances that are solved already during the refinement phase of their counterexample-guided abstraction refinement algorithm.

In planning tasks with SDAC, an optimal plan in the abstraction is not necessarily optimal in the concrete planning task, because the approximation of action costs as the minimum over all concrete states that are contained within an abstract state is an additional source of error. Unlike potential errors due to the abstraction it is not resolved when a plan is found that can be applied in both transition systems.

Example 5. *Consider the abstract transition system of a planning task with SDAC with propositional variables x and y depicted in the following figure.*



There are the actions $a_1 = \langle \top, x \wedge y \rangle$ and $a_2 = \langle \top, \neg x \wedge y \rangle$ with cost functions $c_{a_1} = 2x + 1$ and $c_{a_2} = 1$; initial state $s_0 = 10$; and goal condition $x \wedge y$. The figure above omits transitions that are self-loops or start in the goal state. Since all abstract costs are 1, the optimal abstract plan is $\pi_1 = \langle a_1 \rangle$ with abstract cost $\mathrm{cost}^\alpha(\pi_1) = 1$, and π_1 is also a

concrete plan with cost $cost(\pi_1) = 3$. *However, the optimal concrete plan is* $\pi_2 = \langle a_2, a_1 \rangle$ *with concrete and abstract cost* $cost(\pi_2) = cost^\alpha(\pi_2) = 2$.

For planning tasks with SDAC, optimal abstract plans must therefore satisfy an additional constraint to be *optimal* concrete plans: we not only require that the abstract plan is concretizable, but also that its cost is identical in both the abstract and the concrete transition system.

Definition 8. *Let* α *be an abstraction mapping and* π *an optimal abstract plan. We call* π concretizable at the same cost *if* π *is concretizable and* $cost^\alpha(\pi) = cost(\pi)$.

As the cost of each abstract plan is a lower bound on its concrete cost, an optimal abstract plan that is *concretizable at the same cost* must be optimal in the concrete planning task as well.

Proposition 5. *Let* π *be an optimal abstract plan in* $\mathcal{T}^\alpha$ *that is concretizable at the same cost. Then* π *is an optimal plan in the concrete transition system* $\mathcal{T}$. □

This will become relevant below when we generalize counterexample guided abstraction refinement (CEGAR) to SDAC. Besides the usual types of flaws in spurious abstract plans, we define *cost mismatch flaws* that occur if a plan may be concretizable, but not at the same cost.

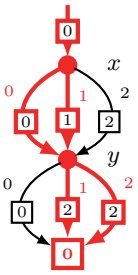
Generalized CEGAR

Counterexample guided abstraction refinement is an established method to derive Cartesian abstractions. It has been introduced first in the model checking community (Clarke et al. 2000), and has recently been applied to classical planning by Seipp and Helmert (2013). The algorithm maintains an explicit abstraction which is initialized as the trivial abstraction with a single abstract state. The abstract transition system is refined iteratively until a termination criterion is met or until an optimal abstract plan that is concretizable is found. In the former case, the resulting transition system can be used to derive admissible heuristic estimates to guide the search procedure, while the abstract plan is also an optimal plan for the concrete task in the latter.

In each iteration, CEGAR aims to compute an abstract plan. If it fails, the problem is unsolvable. Otherwise, if the plan is not concretizable, it searches for a *flaw* in the concrete execution of that plan and refines the corresponding abstract state by splitting it. Seipp and Helmert describe three types of flaws that can be encountered: first, a concrete state is not contained in the corresponding abstract state; second, an action precondition is not satisfied; and third, the abstract plan does not finish in a goal state of the concrete task.

The basic procedure of the algorithm remains untouched even in the presence of SDAC. However, the additional constraint on optimal abstract plans that are also optimal concrete plans implies that we also have to generalize the algorithm to SDAC by adding a fourth kind of flaw: a *cost mismatch flaw* is encountered if the application of an action incurs different costs in the abstract transition system and the concrete one. In our implementation of the *generalized CEGAR* algorithm, if a cost mismatch flaw is encountered in a concrete state s under application of action a, we split the abstract state $\hat{s} = \alpha(s)$ into two parts $\hat{s}_1$ and $\hat{s}_2$ by selecting a variable v randomly among all variables with (i) $v \in supp(a)$, and (ii) there are $d, d' \in \hat{s}(v)$ that contribute different partial costs in the evaluation of $c_a^\alpha(\hat{s})$. With $d = s(v)$, we create $\hat{s}_1$ with $\hat{s}_1(v) = \{d\}$ and $\hat{s}_1(v') = \hat{s}(v')$ for all $v' \neq v$, and $\hat{s}_2$ with $\hat{s}_2(v) = \hat{s}(v) \setminus \{d\}$ and $\hat{s}_2(v') = \hat{s}(v')$ for all $v' \neq v$. 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 an example.

Example 6. *Consider an action* a *with the cost function that is encoded by the depicted EVMDD for two variables* x *and* y *with* $\mathcal{D}_x = \mathcal{D}_y = \{0, 1, 2\}$, *and let* $\hat{s} = \{0, 1\} \times \{1, 2\}$ *be the abstract state that is indicated by the highlighted edges. Both* x *and* y *satisfy condition (i), but only* x *satisfies condition (ii) since both partial costs incurred by* y *are* 2. *This can also be observed from the costs of applying* a *in the contained concrete states, which are* $c_a(01) = c_a(02) = 2$ *and* $c_a(11) = c_a(12) = 3$ *and hence independent of variable* y.



This guarantees that generalized CEGAR only terminates if the termination criterion is met or when an optimal abstract plan is found that is concretizable at the same cost. Hence, our version of CEGAR is a generalization of the algorithm of Seipp and Helmert (2013), i.e., it behaves identically when applied to a classical planning task but is able to deal with state-dependent action costs. Finally, we want to clarify why the worst-case exponentiality of computing abstract cost values is problematic in the first place if we build the whole abstract state space explicitly anyway. This is because the computation of abstract cost values can be "more exponential" than the computation of the abstract state space; since cost functions depend on variables that get abstracted away, we have to minimize over all (partial) states on which the function depends, not only over all abstract states.

Experimental Evaluation

We evaluate our approach on domains from the benchmark set of the last two International Probabilistic Planning Competitions (IPPC), which are the only benchmarks we are aware of that include state-dependent action costs (albeit in the form of rewards). However, the overhead required to adapt our abstraction heuristic to the *probabilistic* setting would distract from the main purpose of this work. Hence, we consider *deterministic* variants of the IPPC domains that are modified as follows: first, we use the most-likely determinization of the probabilistic task where the most likely action outcome replaces all probabilistic effects. Second, we transform the reward function to a non-negative cost function by setting $c_a(s) = R_a^{\max} - R_a(s)$, where $R_a^{\max}$ is the highest reward that is possible under application of action a, and $R_a(s)$ is its reward function. Third, since all benchmark instances are *finite-horizon* problems with the objective of maximizing the expected reward over a finite number

of steps, we determine a set of goal facts by computing all states with an applicable zero-cost action.

In all domains but GAME OF LIFE, this leads to an intuitive description of the goal. However, for ELEVATORS, SYSADMIN and TAMARISK the goal either holds in the initial state or is trivial to achieve. We therefore change the initial state in these domains in a way that the goal is maximally hard to achieve (i. e., passengers wait on all floors and are in all elevators in ELEVATORS, all computers are switched off in SYSADMIN and there is a tamarisk in each slot in TAMARISK). In ACADEMIC ADVISING, NAVIGATION, SKILL TEACHING and TRIANGLE TIREWORLD, the initial state is already significant. Besides GAME OF LIFE, we also omit WILDFIRE, RECON, and CROSSING TRAFFIC in our evaluation because all instances are either unsolvable or trivial. The last existing domain, TRAFFIC, is omitted because the set of goal states is too large for our conversion procedure to handle in the majority of instances. This results in seven domains with ten instances each that are used in our empirical evaluation.

Our abstraction heuristic is based on the generalized CEGAR approach that has been described in the previous section. It has several important properties: following Proposition 4, it computes admissible heuristic estimates; it is an *anytime* heuristic in the sense that it improves with increasing computation time; and it requires a *precomputation step* where either a solution is found (compare to Proposition 5) or a lookup table is created, which allows low heuristic "computation" times during search.

We compare the heuristic that is computed with CEGAR, h^{cegar}, with two other heuristics that support SDAC: the first heuristic, h^{ids}, is based on iterative deepening search and is used successfully in the PROST planner (Keller and Eyerich 2012), the winner of the last two IPPCs. In each iteration, it returns the cost of the cheapest possible sequence of actions for the current depth and terminates when a goal state is encountered. The h^{ids} heuristic is an anytime approach which is often remarkably fast, since results can often be computed based on the cached results of previous iterations. It is, however, only admissible in planning tasks where the shortest plan is also the optimal one (ACADEMIC ADVISING, SYSADMIN, NAVIGATION and TRIANGLE TIREWORLD). The other considered heuristic is a variant of the additive heuristic h^{add} that also makes use of EVMDDs to deal with SDAC. Geißer, Keller, and Mattmüller (2015) have recently applied it successfully to ACADEMIC ADVISING. It is neither admissible nor anytime, though. To allow an unbiased comparison, all three heuristics and the used search algorithms have been implemented in the PROST planning framework, which uses the Meddly library (Badar and Miner 2011) to generate EVMDDs.

In the first part of our evaluation, we are interested in the anytime behaviour of h^{cegar} and h^{ids}, i.e., we analyze empirically how the heuristic accuracy evolves with increasing computation time. We only consider the initial states in this setup, and run both heuristics with a time limit of 30 minutes across all instances. Whenever the computed heuristic estimate improves, we report the time and heuristic value. The results for the hardest instance of each domain, which

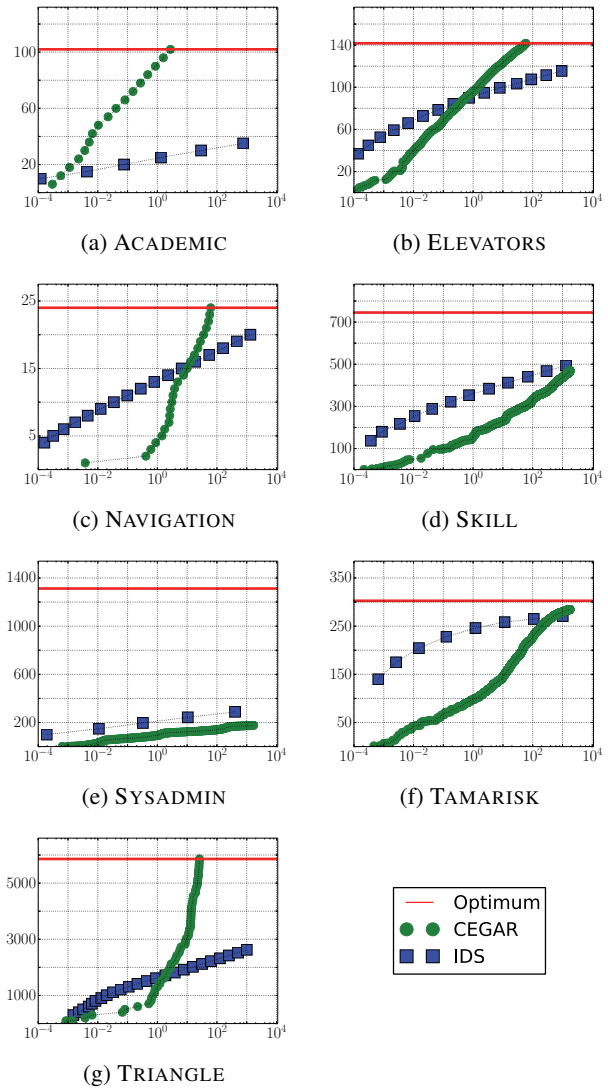


Figure 1: Cost estimate as a function of computation time of h^{cegar} and h^{ids} in the hardest instances of IPPC domains

proved to be representative over all instances, are depicted in Figure 1, along with the cost of the optimal plan which is derived by running A* with h^{cegar} (apart from SYSADMIN, which is computed manually).

It can be seen that h^{cegar} eventually dominates in ACADEMIC ADVISING, ELEVATORS, NAVIGATION, and TRIANGLE TIREWORLD where it improves over h^{ids} after less than ten seconds and solves all instances in less than a minute. While this might seem like much for a heuristic, one has to keep in mind that h^{cegar} – unlike h^{ids} – performs this computation only once initially, and merely looks up precomputed results during search. With this in mind, the results for SKILL TEACHING and TAMARISK also look better than at first glance. The final domain, SYSADMIN, is the hardest for both heuristics. It is also the only domain with a significant number of instances that cannot be solved within

	ACAD	ELEV	NAV	SKILL	SYS	TAMA	TRIA
h^{cegar} (1800s)	**1.00**	**1.00**	**1.00**	0.62	0.13	0.94	**1.00**
h^{cegar} (10s)	**1.00**	**1.00**	0.62	0.29	0.08	0.50	0.41
h^{cegar} (1s)	0.79	0.68	0.17	0.21	0.07	0.32	0.22
h^{add}	2.82	3.90	**1.00**	11.21	1.93	12.53	0.17

Table 1: Relative accuracy of heuristic estimates compared to optimal costs for h^{cegar} with varying timeouts and h^{add}

our time limit of 30 minutes – apart from the eight unsolved SYSADMIN instances, there are only two from the ELEVATORS domain, three from SKILL TEACHING and a single TAMARISK instance. Figure 2 shows the time that is required by the heuristics to compute optimal estimates. With h^{cegar}, 56 out of 70 instances are solved optimally within the given time limit, while h^{ids} only solves 29. Moreover, apart from some trivial instances that are solved by both approaches in less than a second, h^{cegar} solves most instances significantly faster than h^{ids}.

As a final remark on our first experiment, let us also point out that we have even used CEGAR to rule out the unsolvable instances from the CROSSING TRAFFIC and WILDFIRE domains (which led to their exclusion from the experiment), which it is able to detect fairly quickly. The h^{ids} heuristic, on the other hand, can only detect that an instance is unsolvable if it explores the whole state space.

In our second experiment, we compare h^{cegar} to the variant of the additive heuristic that is able to deal with state-dependent action costs. As h^{add} is neither admissible nor does it improve with increasing computation time, we compare the accuracy of the heuristics relative to the perfect heuristic. The results for the hardest instances of our evaluation domains for h^{cegar} with three different timeout settings (1800, 10, and 1 second) and the additive heuristic are depicted in Table 1. If an entry is smaller than 1.0, costs are underestimated, while values higher than 1.0 denote overestimations. Apart from TRIANGLE TIREWORLD (where the delete relaxation allows the use of a single spare tire arbitrarily often), h^{add} overestimates the cost in all tasks. The admissible CEGAR heuristic converges towards the perfect heuristic from below. If provided with a timeout of 30 minutes, it clearly outperforms h^{add} on all domains but SYSADMIN in terms of accuracy. With lower timeout, the picture is less clear, but it is again the case that h^{cegar} needs to be computed only once while h^{add} is computed in every state, such that higher precomputation times can be justified by the time that is saved during search.

Conclusion

In this paper, we studied abstractions for planning with state-dependent action costs. They can be used to derive admissible heuristics required for optimal planning. We identified the efficient computation of abstract transition costs as the main challenge, and showed that this challenge can be solved for abstractions that are at most as fine-grained as Cartesian ones, but generally not for finer abstractions. The main idea is that the independence between state variables

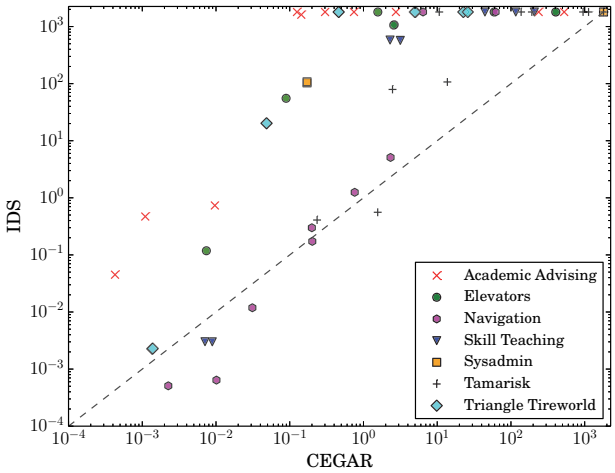


Figure 2: Time in seconds to reach optimal cost estimates with h^{cegar} and h^{ids} in the considered IPPC domains

in Cartesian abstractions can be exploited when computing *abstract* cost values based on an EVMDD encoding the *concrete* cost function.

Based on this observation we generalized CEGAR, a standard technique to come up with Cartesian abstractions, to state-dependent action costs. Besides the usual flaws detected by CEGAR in spurious solutions, we identified cost mismatch flaws that occur if an abstract plan is already concretizable but still too cheap. We therefore presented a method to resolve such flaws in our generalized CEGAR algorithm. Our experimental evaluation on a set of determinized IPPC benchmarks shows that our abstraction heuristic has a higher accuracy than other heuristics that support state-dependent action costs.

Since our heuristic is admissible, it allows, for the first time, to apply well-known standard planning techniques, like A*, to planning tasks with state-dependent action costs. Moreover, we think that by studying planning with state-dependent action costs we may gain insight into classical planning techniques which alter cost functions, as it is done, for example, in cost partitioning.

Acknowledgments. 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.

References

Badar, J., and Miner, A. 2011. MEDDLY: Multi-terminal and Edge-valued Decision Diagram LibrarY. http://meddly.sourceforge.net/. Accessed: 2015-11-22.

Ball, T.; Podelski, A.; and Rajamani, S. K. 2001. Boolean and cartesian abstraction for model checking C programs. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, 268–283.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Pro-*

ceedings of the 14th National Conference on Artificial Intelligence (AAAI 1997), 714–719.

Ciardo, G., and Siminiceanu, R. 2002. 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)*, 256–273.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided abstraction refinement. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, 154–169.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *Preproceedings of the 6th European Conference on Planning (ECP 2001)*, 13–24.

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

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete relaxations for planning with state-dependent action costs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1573–1579.

Hernádvölgyi, I. T., and Holte, R. C. 2000. Experiments with automatically created memory-based heuristics. In *Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation (SARA 2000)*, 281–290.

Ivankovic, F.; Haslum, P.; Thiébaux, S.; Shivashankar, V.; and Nau, D. S. 2014. Optimal planning with global numerical state constraints. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 119–127.

Lai, Y.-T.; Pedram, M.; and Vrudhula, S. B. K. 1996. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers* 45(2):247–255.

Seipp, J., and Helmert, M. 2013. Counterexample-guided cartesian abstraction refinement. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 347–351.

van den Briel, M.; Kambhampati, S.; and Vossen, T. 2007. Fluent merging: A general technique to improve reachability heuristics and factored planning. In *Proceedings of the ICAPS 2007 Workshop on Heuristics for Domain-Independent Planning (HDIP 2007)*.