On the Compilability and Expressive Power of State-Dependent Action Costs

David Speck, David Borukhson, Robert Mattmüller, Bernhard Nebel
University of Freiburg, Germany
⟨speckd, borukhd, mattmuel, nebel⟩@informatik.uni-freiburg.de

Abstract

While state-dependent action costs are practically relevant and have been studied before, it is still unclear if they are an essential feature of planning tasks. In this paper, we study to what extent state-dependent action costs are an essential feature by analyzing under which circumstances they can be compiled away. We give a comprehensive classification for combinations of action cost functions and possible cost measures for the compilations.

Our theoretical results show that if both task sizes and plan lengths are to be preserved polynomially, then the boundary between compilability and non-compilability lies between FP and FPSPACE computable action cost functions (under a mild assumption on the polynomial hierarchy). Preserving task sizes polynomially and plan lengths linearly at the same time is impossible.

		Desired plan length preservation		
		linearly	polynomially	don't care
GCF complexity	FP	*impossible* (Thm. 5)	*possible* using DTM compilation (Thm. 3)	
	FPSPACE	*impossible* (unless PH collapses at the third level) (Thm. 4)		*possible* using DTM compilation (Thm. 2)

Table 1: Existence results for compilation schemes preserving task sizes polynomially depending on the computational complexity of the generic cost functions (rows) and the desired plan length preservation (columns).

Introduction

A planning task is said to have state-dependent action costs (Geißer 2018; Ivankovic, Gordon, and Haslum 2019) if the cost of one or more of its actions depends on the state where the action is applied. State-dependent action costs occur naturally, e. g., in numeric planning or in probabilistic planning in the form of state-dependent rewards of a Markov decision process (MDP).

Depending on the planning algorithm used, state-dependent action costs can be handled "natively", e. g., during uninformed explicit-state forward search, where they are only needed to compute g-values of successor states. For other algorithms, such as the computation of many goal-distance heuristics, compiling state-dependent action costs away into planning tasks with constant, i. e., state-independent costs only, is required. Corraya et al. (2019) showed that simply ignoring that costs are state-dependent can not only make the solution quality exponentially worse, but also affect search performance. Recent work (Geißer, Keller, and Mattmüller 2015, 2016; Geißer 2018) has studied representations of state-dependent action costs as decision diagrams that are compact if additive structure in cost functions can be exploited. Based on such representations, translations were introduced, and the extent to which such a representation can be used to compute informative goal-distance heuristics was investigated.

The EVMDD-based translation of state-dependent action costs (Geißer, Keller, and Mattmüller 2015) represents action cost functions as edge-valued multi-valued decision diagrams (EVMDDs) (Ciardo and Siminiceanu 2002) and turns each edge of those diagrams into an auxiliary action. This translation was shown to be "heuristic friendly" for a range of heuristics (Geißer 2018), while leading only to a polynomial increase in representation size for many cost functions, such as weighted sums of state features. In the worst case, however, this translation can increase the size of a planning task exponentially. Similarly, the combinatorial translation that introduces a new action for every combination of an original action with a partial assignment to the state variables relevant to that action's costs necessarily increases task sizes exponentially.

This observed worst-case exponential increase raises the question whether this is generally unavoidable, i. e., if or to what extent state-dependent action costs are an essential feature of planning tasks and to what extent they are syntactic sugar. This is the question we address in this paper. It is closely related to an earlier study of the compilability of conditional effects (Nebel 2000), another state-dependent feature of planning tasks. There, it was shown that conditional effects cannot always be compiled away with only polynomial overhead.

We use the notion of compilability (Nebel 2000) to formalize an essential feature. We already know from the EVMDD-based and the combinatorial translations that it is possible to preserve plan lengths polynomially or even exactly, respectively, at the expense of exponential increases in task size. Note that according to the definition of Nebel (2000), a compilation may only increase the task size polynomially, which means that, strictly speaking, all existing translations (EVMDD-based and combinatorial) are no compilations. In this paper, however, we focus on translations that preserve task sizes polynomially, i. e., compilations, and draw the boundary between cases in which plan lengths can also be preserved polynomially and cases in which this is impossible. Our results show that this depends on the expressiveness of the cost functions that are allowed (Table 1). For planning tasks in which all action cost functions can be computed in FPSPACE, state-dependent action costs can be compiled away into actions with constant costs while preserving the size of the planning task polynomially. However, additional polynomial preservation of plan lengths can only be guaranteed if the action cost functions can be computed in FP. This is shown by representing each action cost function as a Turing machine computing it. Then, encoding the simulation of that Turing machine as part of the new planning task constitutes a compilation that preserves task size (and plan lengths in case of FP computable cost functions) polynomially. On the other hand, we can show that preserving both task sizes and plan lengths polynomially is impossible for FPSPACE action cost functions unless the polynomial hierarchy collapses at the third level.

The rest of the paper is structured as follows: We introduce required preliminaries and show that planning with state-dependent action costs is PSPACE-complete, assuming a cost function in FPSPACE, followed by the positive results based on Turing machine compilations, and the impossibility results complementing them.

Preliminaries

First, we define planning tasks and domains with state-dependent action costs (SDAC), and as a special case thereof, tasks and domains with constant action costs (CAC).

Definition 1 (SDAC and CAC planning domain)**.** *An* SDAC planning domain *is a tuple* $\Xi = \langle P, A, C\rangle$*.* $P = \{p_0, \dots, p_{n-1}\}$ *is a finite set of* n propositional atoms*. A (partial) assignment* $s : P \to \{0,1\}$ *is called a (partial) state.* S *denotes the set of all (total) states.* A *is a finite set of* actions*. An action* $a \in A$ *is a pair of partial variable assignments* $\langle \mathit{pre}(a), \mathit{eff}(a)\rangle$*. Action* a *is applicable in a state* $s \in S$ *iff* $\mathit{pre}(a)$ *is consistent with* s*. Application of* a *in* s *results in the state* $s' = s[a]$ *that is consistent with* $\mathit{eff}(a)$ *and agrees with* s *on the values of all variables from* P *not mentioned in* $\mathit{eff}(a)$*. Finally,* $C : S \times A \to \mathbb{N}_0$ *is the* cost function *of* Ξ.

Ξ *is a* CAC planning domain *if* C *is constant in the states, i. e., if* $C(s, a) = C(s', a)$ *for all* $s, s' \in S$ *and* $a \in A$.

When specifying preconditions and effects, we sometimes write $x \doteq b$, with $b \in \{0, 1\}$, for the assignment of value b to variable x. Since we only consider propositional variables, we write x for the fact $x \doteq 1$ and $\neg x$ for the fact $x \doteq 0$. Sometimes we write a (partial) state as a conjunction of literals, e.g., $s = x \wedge \neg x'$, which describes the (partial) state s with $s(x) = 0$ and $s(x') = 1$. With $\mathbf{0}$ we refer to the constant zero function. Indicator functions that return 1 if and only if some condition φ is satisfied, and 0 otherwise, are denoted by $\mathbb{1}_\varphi$.

The size of Ξ is $||\Xi|| = |P| + \sum_{a\in A}(|\mathit{pre}(a)| + |\mathit{eff}(a)|) + ||C||$, where $||C||$ is the encoding size of the action cost function. We assume that C is encoded efficiently in the input in case of SDAC domains. In case of CAC domains, $||C||$ is bounded by $|A| \cdot \lceil \log N \rceil$, where N is the highest action cost value.

Planning domains together with initial states, goal descriptions, and plan cost bounds form planning tasks.

Definition 2 (SDAC and CAC planning task)**.** *An* SDAC planning task *is a tuple* $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G}, \mathbf{B}\rangle$ *consisting of an SDAC* planning domain Ξ*, an initial state* $\mathbf{I}$*, a goal description* $\mathbf{G} \subseteq P$*, and a cost bound* $\mathbf{B} \in \mathbb{N}_0$*. It is a CAC planning task if* Ξ *is a CAC planning domain. The size of* Π *is* $||\Pi|| = ||\Xi|| + |\mathbf{G}| + |\mathbf{I}| + ||\mathbf{B}||$*, where* $||\mathbf{B}||$ *is the binary encoding size of the cost bound.*

Plans for planning tasks are sequences of applicable actions leading from the initial state to a goal state, respecting the given cost bound.

Definition 3 (Plan)**.** *A sequence of actions* $\pi = \langle a_0, \dots, a_{n-1}\rangle$ *is called a* plan *for a planning task* $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G}, \mathbf{B}\rangle$ *if there exist states* $s_0, \dots, s_n$ *such that* $s_0 = \mathbf{I}$*,* $\mathbf{G} \subseteq s_n$*, and* a_i *is applicable in* s_i *and* $s_{i+1} = s_i[a_i]$ *for all* $i = 0, \dots, n-1$*, and* $\sum_{i=0}^{n-1} C(s_i, a_i) \leq \mathbf{B}$.

We call $||\pi|| = n$ the *length* of π.

Note that according to the above definitions of planning domains and tasks, a single domain or task has a fixed set of atoms and actions. What may vary across different planning tasks in one domain are only the initial and goal states, as well as the imposed plan cost bound. In order to study compilability and non-compilability results, it is not sufficient to look at single domains or tasks, though. Rather, we need to investigate entire *families* of planning domains or tasks of increasing size to obtain meaningful results. We refer to such families as *lifted domains*. Since we want action cost functions to be *uniform* across such lifted domains, we define them as *generic cost functions* that are defined for arbitrarily many truth values and action sets.[1]

Definition 4 (Generic cost function)**.** *A* generic cost function *is a deterministic Turing machine that takes as input a finite set of propositional atoms* P *together with a finite set of actions* A *(the* domain context $\langle P, A\rangle$*) together with a state* s *over* P *and an action* $a \in A$*, and that returns a cost value* $n \in \mathbb{N}_0$ *in those domain contexts for which it is defined. For other domain contexts, it terminates right after reading the input, having an undefined output.*

[1]Uniformity here means that a generic cost function can compute only *finitely many*, but possibly more than *one*, different cost functions for different domains within a lifted domain. Since interesting lifted domains will contain *infinitely* many domains, we consider this "uniform enough", especially compared to non-uniform models of computation such as families of Boolean circuits.

This means that we only allow *computable* functions as generic cost functions and rule out pathological ones like cost functions whose value depends, e. g., on whether a Turing machine encoded by the atoms in P halts. A generic cost function is in the function complexity class FX if it computes its result using only resources according to class X.

Notice that a generic cost function $\mathcal{C}$ induces a regular cost function $C : S \times A \to \mathbb{N}_0$ in a given domain context $\langle P, A\rangle$ for which it is defined as $C(s, a) = \mathcal{C}_{P,A}(s, a)$, where S is the set of states over P, and hence also a planning domain $\Xi = \langle P, A, C\rangle$.

Definition 5 (Lifted domain)**.** *A* lifted domain $\mathcal{L} = \langle \mathcal{D}, \mathcal{C}\rangle$ *is a family* $\mathcal{D} = \{(P, A)_i\}_{i\in\mathcal{I}}$ *of domain contexts for some index family* $\mathcal{I}$ *together with a generic cost function* $\mathcal{C}$ *that is defined for all* $(P, A)_i$*,* $i \in \mathcal{I}$*.*

Example 1. *Consider a lifted domain where every task of every domain of that lifted domain has a certain number of objects that can be painted in different colors. Assume that there is an action* a *that colors all red objects blue, and whose cost is the number of affected objects. In a concrete domain* Ξ *with a set* O *containing ten objects and with states encoding, besides other features, the current colors of those objects, the cost function* $C(s, a) = \sum_{o\in O} \mathbb{1}_{s(\mathit{color\text{-}of}(o))=\mathit{red}}$ *is state-dependent, while still restricted to a finite number of objects of interest. The corresponding generic cost function ranges over arbitrarily many objects uniformly. It is linear-time computable.*

The following definition formalizes compilation schemes, which translate from one planning formalism to another while preserving plan existence and polynomially preserving task sizes. We are interested in compilation schemes from planning with SDAC to planning with CAC.

Definition 6 (Compilation scheme)**.** *A compilation scheme or, in short, a compilation is a tuple of functions* $\mathbf{f} = \langle f_\xi, f_\iota, f_g\rangle$ *on planning domains that induces a function on planning tasks as follows:*

$$F(\Pi) = \langle f_\xi(\Xi), \mathbf{I} \cup f_\iota(\Xi), \mathbf{G} \cup f_g(\Xi), \mathbf{B}\rangle,$$

where $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G}, \mathbf{B}\rangle$*, satisfying the following conditions:*

1. *there exists a plan for* $F(\Pi)$ *iff there exists a plan for* Π*,*
2. *the size of the results of* f_ξ*,* f_ι*, and* f_g *is polynomial in the size of the arguments.*

Note that by not allowing to change the cost bound we make sure that for each plan in the original task there must be one with at most cost $\mathbf{B}$ in the target task and vice versa.

Besides preserving planning task sizes polynomially, another desirable property is preservation of plan lengths. This is captured by the following definition.

Definition 7 (Compilations preserving plan length)**.** *A compilation scheme* $\mathbf{f}$ *is said to* preserve plan length exactly *(modulo an additive constant) if for every plan* π *solving an instance* Π*, there exists a plan* π' *solving* $F(\Pi)$ *with* $||\pi'|| \leq ||\pi|| + k$ *for some positive integer constant* k*. It is said to* preserve plan length linearly *if* $||\pi'|| \leq c \cdot ||\pi|| + k$ *for positive integer constants* c *and* k*, and to* preserve plan length polynomially *if* $||\pi'|| \leq p(||\pi||, ||\Pi||)$ *for some polynomial* p*.*

For the compilability results, we make heavy use of Turing machines. The definition below determines the notation we use for them.

Definition 8 (Turing machine)**.** *A (nondeterministic) Turing machine (TM or NTM) is a tuple* $\mathcal{T} = \langle Q, \Gamma, \Sigma, q_0, q_\mathrm{f}, \delta\rangle$ *with the following components: (a)* Q *are the* states *of* $\mathcal{T}$*, (b)* $\Gamma = \{0, 1, \sqcup\}$ *is the* symbol alphabet*, with* $\sqcup$ *being the blank symbol, (c)* $\Sigma = \{0, 1\}$ *are the* input symbols*, (d)* $q_0 \in Q$ *is the* initial state*, (e)* $q_\mathrm{f} \in Q$ *is the* final state*, and (f)* $\delta \subseteq (Q \setminus \{q_\mathrm{f}\} \times \Gamma) \times (Q \times \Gamma \times D)$ *is the* transition relation*, where* $D = \{-1, +1\}$ *represents the head movement in a transition. We call a TM deterministic (DTM) if the transition relation is functional, i.e.,* $\delta : (Q \setminus \{q_\mathrm{f}\}) \times \Gamma \to Q \times \Gamma \times D$*. A TM accepts an input string iff at least one of the possible computational paths starting from that string puts the machine into the final state.*

Let $\mathcal{T}$ be a deterministic Turing machine. By $f_\mathcal{T} : \{0, 1\}^* \to \mathbb{N}_0$ we refer to the *function computed by* $\mathcal{T}$, i. e., $f_\mathcal{T}(v_0 \ldots v_{n-1}) = N$ means that $\mathcal{T}$, starting in the configuration with input string $v_0 \ldots v_{n-1}$ on the tape, terminates with the binary encoding of N on the tape. W.l.o.g, we assume that binary numbers are written from left to right (little-endian) on the output tape and padded with zeros. For example, considering 4 tape cells, the binary number 1010 encodes the value $1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 = 5$. The *space required of* $\mathcal{T}$ is given by $k_\mathcal{T} : \mathbb{N}_0 \to \mathbb{N}$, i. e., $k_\mathcal{T}(n)$ is the number of tape cells used at least once during any execution of $\mathcal{T}$ on any input of length n, and the *time required by* $\mathcal{T}$ is given by $d_\mathcal{T} : \mathbb{N}_0 \to \mathbb{N}_0$, i. e., $d_\mathcal{T}(n)$ is the number of transitions required until reaching a final state during any execution of $\mathcal{T}$ on any input of length n.

Computational Complexity

The first natural question is, of course, whether planning with SDAC is computationally harder than planning with CAC. More precisely, the *bounded plan existence problem of planning with SDAC* is the problem of deciding for a given planning task Π with cost bound $\mathbf{B}$ whether there is a plan that costs $\mathbf{B}$ or less. Interestingly, this question has not yet been answered in the literature, which stems from the fact that it is important to take into account the complexity of the cost function considered, something that has never been done explicitly before. For this purpose it is natural to consider that the cost function is not harder than the plan existence problem when planning itself, i.e. in FPSPACE. It turns out that the *bounded plan existence problem of planning with SDAC* is PSPACE-complete just like ordinary classical planning (Theorem 1).

Theorem 1. *Bounded plan existence of planning with SDAC is* PSPACE*-complete, provided the cost function is in* FPSPACE*.*

Proof. PSPACE-hardness results from reducing the well-known bounded plan existence problem (with unit costs) (Bylander 1994) to our problem, where the plan length corresponds to the plan cost. PSPACE-membership can be proven by defining a nondeterministic TM that guesses an action for application in each step, evaluating the applied actions and adding up their costs. This nondeterministic TM terminates if the selected action is not applicable, or the given cost bound $\mathbf{B}$ is exceeded. Since only the current state, the summed up costs and the computation of the selected cost function (in FPSPACE) must be maintained at any time, this TM is in NPSPACE, which is known to be equivalent to PSPACE (Savitch 1970). □

Possibility Results

As we have shown that planning with SDAC and planning with CAC are in the same complexity class (Theorem 1), the question arises under which circumstances SDAC can be compiled away. In other words, we are interested in the expressive power of SDAC, which is a measure of how concisely planning domains and plans can be expressed in a given formalism with compilation schemes (Nebel 2000). In this section, we define such a compilation scheme that compiles a given planning task with SDAC into a planning task with CAC, and prove that this compilation preserves the plan length polynomially if the generic cost function is computable in polynomial time.

The basic idea of our compilation is to simulate a DTM $\mathcal{T}$, which computes the cost of an action, with planning. Bylander (1994) originally showed how to simulate a given DTM with planning. Similar to Bylander (1994), we define the simulation of a DTM as follows (Definition 9).

Definition 9 (DTM planning simulation). *Let $\mathcal{T} = \langle Q, \Gamma, \Sigma, q_0, q_{\mathrm{f}}, \delta\rangle$ be a $k_{\mathcal{T}}$-space bounded deterministic Turing machine, and $s = v_0 \dots v_{n-1}$ an input of length n. We define $P_{\mathcal{T}}$ as the set of propositional state variables that allows encoding a configuration of $\mathcal{T}$, with atomic propositions encoding the current internal state of the Turing machine, its head position*[2]*, and the content of every cell that can be reached in the given space bound, as follows.*

$$\begin{aligned} P_{\mathcal{T}} = {} & \{\mathit{state}_q \mid q \in Q\} \\ & \cup \{\mathit{head}_i \mid -1 \le i \le k_{\mathcal{T}}(n)\} \\ & \cup \{\mathit{content}_{i,c} \mid 0 \le i < k_{\mathcal{T}}(n), c \in \Gamma\} \end{aligned}$$

In order to produce the initial Turing machine configuration with input string $s = v_0 \dots v_{n-1}$ on the tape, initial Turing machine state q_0, and head on the left, we define $\mathbf{I}_{\mathcal{T}}(s) : P_{\mathcal{T}} \mapsto \{0, 1\}$ as a function, assigning each variable $P_{\mathcal{T}}$ a truth value as follows.

$$\mathbf{I}_{\mathcal{T}}(s)(x) = \begin{cases} 1 & \text{if } x \in \{\mathit{state}_{q_0}, \mathit{head}_0\} \\ 1 & \text{if } x \in \{\mathit{content}_{i,v_i} \mid 0 \le i < n\} \\ 1 & \text{if } x \in \{\mathit{content}_{i,\llcorner\!\lrcorner} \mid n \le i < k_{\mathcal{T}}(n)\} \\ 0 & \text{otherwise} \end{cases}$$

Finally, we define a set of actions $A_{\mathcal{T}}$ which "simulates" the transitions of $\mathcal{T}$ as follows.

$$\begin{aligned} A_{\mathcal{T}} = \{ & a_{q,c,q',c',d,i} \text{ for each transition } \delta(q,c) = \langle q', c', d\rangle \\ & \text{and each cell position } 0 \le i < k_{\mathcal{T}}(n)\} \\ \mathit{pre}(& a_{q,c,q',c',d,i}) = \mathit{state}_q \wedge \mathit{head}_i \wedge \mathit{content}_{i,c} \\ \mathit{eff}(& a_{q,c,q',c',d,i}) = \neg\mathit{head}_i \wedge \mathit{head}_{i+d} \\ & \dots \wedge \neg\mathit{state}_q \wedge \mathit{state}_{q'} \qquad (\text{only if } q \ne q') \\ & \dots \wedge \neg\mathit{content}_{i,c} \wedge \mathit{content}_{i,c'} \quad (\text{only if } c \ne c') \end{aligned}$$

By construction, the components of Definition 9 define a planning task that simulates a given DTM $\mathcal{T}$. More precisely, for a given DTM $\mathcal{T} = \langle Q, \Gamma, \Sigma, q_0, q_{\mathrm{f}}, \delta\rangle$ with input $s = v_0 \dots v_{n-1}$, the planning task $\Pi_{\mathcal{T}} = \langle \Xi, \mathbf{I}_{\mathcal{T}}(s), q_{\mathrm{f}}, 0\rangle$ with $\Xi = \langle P_{\mathcal{T}}, A_{\mathcal{T}}, \mathbf{0}\rangle$ simulates the behavior of $\mathcal{T}$. In other words, after executing the unique plan π for $\Pi_{\mathcal{T}}$, the variables $\mathit{content}_{i,c} \in P_{\mathcal{T}}$ with $0 \le i < k_{\mathcal{T}}(n)$ and $c \in \{0, 1\}$ encode the function $f_{\mathcal{T}}(s)$ computed by $\mathcal{T}$ in binary. Note that, by construction, for each $0 \le i < k_{\mathcal{T}}(n)$, either $\mathit{content}_{i,0}$ or $\mathit{content}_{i,1}$ is true after executing π.

Since we assume that all (generic) action cost functions are computable, we know that for every action a there exists a DTM $\mathcal{T}_a$ that computes the function $f_{\mathcal{T}_a}(s) = C(s, a)$ for all states s. We further assume that $\mathcal{T}_a$ is space-bounded polynomially (or more tightly). In the following, we choose such a DTM $\mathcal{T}_a$ that computes this function asymptotically optimally. Informally, this means that for large inputs, $\mathcal{T}_a$ will at worst perform a constant factor (regardless of input size) worse than the best possible DTM. This is necessary to ensure that $\mathcal{T}_a$ maintains the same complexity as the generic cost function.

Next, we define and explain a translation called $\mathbf{f}^{\mathrm{SDAC}}$ that transforms a given planning task with SDAC into a planning task with CAC (Definition 10).

Definition 10 (SDAC translation). *Let $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G}, \mathbf{B}\rangle$ be a an* SDAC *planning task with $\Xi = \langle P, A, C\rangle$. We define the* SDAC translation *as the tuple of functions $\mathbf{f}^{SDAC} = \langle f^{SDAC}_{\xi}, f^{SDAC}_{\iota}, f^{SDAC}_{g}\rangle$, which are defined and described in detail below and which yield the new planning task $F^{SDAC}(\Pi)$.*

In the following we define and describe the functioning of the SDAC translation $\mathbf{f}^{\mathrm{SDAC}}$ (Definition 10) and assume $N_m = \{0, \dots, m\}$. We first define the new set of propositional variables P', the new set of actions A' and the new cost function C', which result from f^{SDAC}_{ξ}, i.e., $f^{\mathrm{SDAC}}_{\xi}(\Xi) = \langle P', A', C'\rangle$.

$$\begin{aligned} P' = P \cup & \bigcup_{a \in A} P_{\mathcal{T}_a} \\ & \cup \{\mathrm{idle}\} && (1) \\ & \cup \{\mathrm{starting}^a \mid a \in A\} && (2) \\ & \cup \{\mathrm{copyingToTape}^a_i, \mid a \in A, i \in N_{|P|}\} && (3) \\ & \cup \{\mathrm{simulatingDTM}^a \mid a \in A\} && (4) \\ & \cup \{\mathrm{readingFromTape}^a_i, \mid a \in A, i \in N_{k_{\mathcal{T}_a}(|P|)}\} && (5) \end{aligned}$$

The new propositions P' include all propositions $P_{\mathcal{T}_a}$ needed to encode the configurations of all cost Turing machines $\mathcal{T}_a$ as well as additional propositions that encode

[2]For simplicity, without loss of generality, we use DTMs that never move to the left of the initial head position.

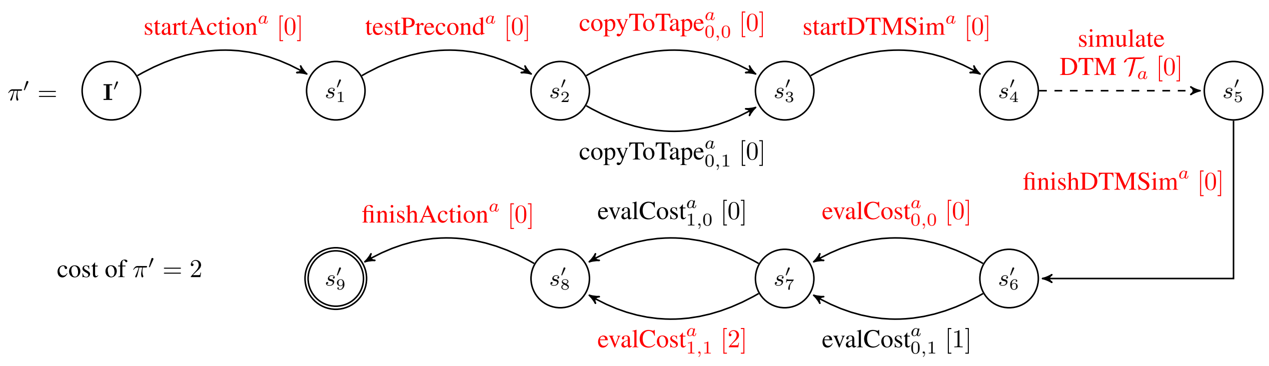

Figure 1: Visualization of the plan π' resulting from the $\mathbf{f}^{\text{SDAC}}$ translation for the task $\Pi' = F^{\text{SDAC}}(\Pi)$ of Example 2. This plan corresponds to the original plan π for Π consisting of a single action a. The red actions are part of π' and the action cost of each action is shown in the square brackets. The black actions are not applicable by construction.

mutually exclusive phases of the translation of all actions a: (1) an idle phase when no action is currently selected; (2) a phase starting^a when action a has been selected as the next action to execute; (3) a copying phase during which the original propositions of the planning task are copied onto the Turing machine tape (technically: into new propositions representing the Turing machine tape content, called copyingToTape^a_i for each original proposition index i); (4) a phase during which the Turing machine $\mathcal{T}_a$ is simulated, called simulatingDTM^a; and (5) a phase during which the cost value computed by $\mathcal{T}_a$ is read from the Turing machine tape (technically: from the new propositions encoding the tape content) bit by bit, and corresponding partial costs for the active bits are incurred in the new planning task $F^{\text{SDAC}}(\Pi)$, called readingFromTape^a_i for each relevant tape cell index i.

The new set of actions A' contains, for each original action a, new actions transitioning between those phases as well as new actions executing the phases.

$$A' = \bigcup_{a \in A} A'_a \text{ with}$$

$$\begin{aligned}
A'_a = \; & \\
& \{\text{startAction}^a\} && (6)\\
\cup\, & \{\text{testPrecond}^a\} && (7)\\
\cup\, & \{\text{copyToTape}^a_{i,1} \mid i \in N_{|P|-1}\} && (8)\\
\cup\, & \{\text{copyToTape}^a_{i,0} \mid i \in N_{|P|-1}\} && (9)\\
\cup\, & \{\text{startDTMSim}^a\} && (10)\\
\cup\, & \{\text{simDTMStep}^a_{a'} \mid a' \in A_{\mathcal{T}_a}\} && (11)\\
\cup\, & \{\text{finishDTMSim}^a\} && (12)\\
\cup\, & \{\text{evalCost}^a_{i,1} \mid i \in N_{k_{\mathcal{T}_a}(|P|)-1}\} && (13)\\
\cup\, & \{\text{evalCost}^a_{i,0} \mid i \in N_{k_{\mathcal{T}_a}(|P|)-1}\} && (14)\\
\cup\, & \{\text{finishAction}^a\} && (15)
\end{aligned}$$

Preconditions and effects of those actions are as follows (we abbreviate content as ct, copyingToTape as cp, simulatingDTM as sim, and readingCostFromTape as rd):

- startAction^a $= \langle \text{idle}, \neg\text{idle} \wedge \text{starting}^a \wedge \textit{reset-dtm}^a \rangle$, where $\textit{reset-dtm}^a \equiv \bigwedge_{x \in P_{\mathcal{T}_a}} x \doteq \mathbf{I}_{\mathcal{T}_a}(\mathbf{0})(x)$.
(Action a is selected as the next action to be executed, a transition from the idle phase to the starting^a phase occurs, and all Turing machine variables of $\mathcal{T}_a$ are reset to their initial values.)
- testPrecond^a $= \langle \text{starting}^a \wedge \textit{pre}(a), \neg\text{starting}^a \wedge \text{cp}^a_0 \rangle$.
(The precondition of a is tested and, if successful, the phase is entered during which the Turing machine tape is filled.)
- copyToTape$^a_{i,1}$ $= \langle \text{cp}^a_i \wedge p_i, \neg\text{cp}^a_i \wedge \text{cp}^a_{i+1} \wedge \text{ct}_{i,1} \wedge \neg\text{ct}_{i,0} \rangle$,
copyToTape$^a_{i,0}$ $= \langle \text{cp}^a_i \wedge \neg p_i, \neg\text{cp}^a_i \wedge \text{cp}^a_{i+1} \wedge \text{ct}_{i,0} \wedge \neg\text{ct}_{i,1} \rangle$.
(The value of p_i is copied to the tape, either as a 1 by the first action, or as a 0 by the second action.)
- startDTMSim^a $= \langle \text{cp}^a_{|P|}, \neg\text{cp}^a_{|P|} \wedge \text{sim}^a \rangle$.
(Once all propositions have been written to the tape, the Turing machine simulation phase is entered.)
- simDTMStep$^a_{a'}$ $= \langle \text{sim}^a \wedge \textit{pre}(a'), \textit{eff}(a') \rangle$.
(The Turing machine step encoded by action a' is simulated, provided we are in the simulation phase.)
- finishDTMSim^a $= \langle \text{sim}^a \wedge \text{state}_{q_f}, \neg\text{sim}^a \wedge \text{rd}^a_0 \rangle$.
(Once the Turing machine simulation has reached the final state q_f of the machine, the simulation terminates and the cost evaluation phase is entered.)
- evalCost$^a_{i,1}$ $= \langle \text{rd}^a_i \wedge \text{ct}_{i,1}, \neg\text{rd}^a_i \wedge \text{rd}^a_{i+1} \rangle$,
evalCost$^a_{i,0}$ $= \langle \text{rd}^a_i \wedge \text{ct}_{i,0}, \neg\text{rd}^a_i \wedge \text{rd}^a_{i+1} \rangle$.
(The value of the i-th least significant bit from the cost value is read from the tape, either as a 1 by the first action, or as a 0 by the second action. Actions evalCost$^a_{i,1}$ are the only actions in the translation with non-zero costs – see below.)
- finishAction^a $= \langle \text{rd}^a_{k_{\mathcal{T}}(|P|)}, \neg\text{rd}^a_{k_{\mathcal{T}}(|P|)} \wedge \textit{eff}(a) \wedge \text{idle} \rangle$.
(Once the cost value has been read from the tape, the effect of the original action a is applied – this has to happen after the cost evaluation, since otherwise the effect could change the values of variables on which the cost of a depends – and the idle phase is entered again.)

The new action cost function C' assigns a cost of zero to all actions except for the evalCost actions that read the computed cost value c from the tape (in binary) and make sure that costs of c are accumulated in the planning task. This is achieved by reading the value bitwise, such that the action $\text{evalCost}^a_{i,1}$ costs 2^i when the bit is in the i-th position and is set to 1. If it is set to 0, it contributes cost 0. Formally,

$$C'(s,a') = \begin{cases} 2^i & \textit{if } a' \in \{\text{evalCost}^a_{i,1} \mid i \in N_{k_{\mathcal{T}_a}(|P|)-1}\} \\ 0 & \textit{otherwise} \end{cases}$$

for all $s \in S, a' \in A'$.

Together, P', A', and C' form the translated domain:

$$f^{\text{SDAC}}_{\xi}(\langle P, A, C\rangle) = \langle P', A', C'\rangle.$$

Finally, f^{SDAC}_{ι} and f^{SDAC}_{g} are defined a follows.

$$f^{\text{SDAC}}_{\iota}(p) = \begin{cases} 1 & \textit{if } p = \text{idle} \\ 0 & \textit{otherwise} \end{cases}$$

$$f^{\text{SDAC}}_{g} = \text{idle}.$$

Before we show that $\mathbf{f}^{\text{SDAC}}$ is indeed a compilation scheme (Theorem 2), i.e., satisfies the two criteria of Definition 6, let us consider Example 2 to get a better understanding of the functioning of $\mathbf{f}^{\text{SDAC}}$.

Example 2. *Consider a simple SDAC planning task* $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G}, 2\rangle$ *with one propositional variable* p_0*, one action* $a = \langle \neg p_0, p_0\rangle$ *and cost function* $C(s,a) = 2 \cdot \mathbb{1}_{p_0 \doteq 0}$*, i.e.,* $\Xi = \langle \{p_0\}, \{a\}, C\rangle$*. We assume that* $\mathcal{T}_a$ *computes the function* $f_{\mathcal{T}_a}(s) = C(s,a)$ *for all states* s *requiring two tape cells. Furthermore,* Π *has the initial state* $\mathbf{I}(p_0) = 0$ *and the goal description* $\mathbf{G} = p_0$*. Clearly, the unique plan for* Π *is* $\pi = \langle a\rangle$ *with cost* 2*.*

Let us now consider the task $\Pi' = F^{\text{SDAC}}(\Pi)$ *induced by the SDAC translation* $\mathbf{f}^{\text{SDAC}}$ *(Definition 10). The unique plan* π' *for* Π' *is visualized in Figure 1. In the initial state* $\mathbf{I}'$ *of* Π'*, all variables are false except idle. Thus, the only initially applicable action is startAction*^{a}*, leading to state* s'_1*. After that, the action testPrecond*^{a} *is applicable, since the original precondition* $pre(a)$ *holds in* s'_1*. Now the value of* p_0 *is written to the tape of the DTM* $\mathcal{T}_a$ *which we simulate within the planning task. Since* p_0 *is false, only the copyToTape*$^a_{0,0}$ *action is applicable. The next actions start, execute, and terminate the simulation of the DTM* $\mathcal{T}_a$*, which computes the cost of action* a *with respect to the current state* ($p_0 \doteq 0$)*. Now the cost evaluation begins, which reads the output of* $\mathcal{T}_a$ *from the new propositions representing the tape content of the Turing machine. The cost of applying action* a *in* ($p_0 \doteq 0$) *is two, which is why the output tape reads* 01 *in binary. Therefore, the first evaluation action costs* $0 \cdot 2^0 = 0$ *and the second costs* $1 \cdot 2^1 = 2$*. Finally, action* a *is finished, which leads us to the goal state. The only action with non-zero cost was the evaluation action with a cost of 2, which is why the final plan* π' *for* Π' *has a cost of 2.*

Theorem 2. *For tasks from a lifted domain* $\mathcal{L} = \langle \mathcal{D}, \mathcal{C}\rangle$ *with a generic cost function* $\mathcal{C}$ *in* FPSPACE*, the SDAC translation* $\mathbf{f}^{\text{SDAC}}$ *is a compilation scheme from SDAC tasks to CAC tasks.*

Proof. We have to show that $\mathbf{f}^{\text{SDAC}}$ preserves bounded plan existence, that f^{SDAC}_{ξ}, f^{SDAC}_{ι}, and f^{SDAC}_{g} are of polynomial size, and that the resulting tasks are CAC tasks. Plan existence (ignoring plan costs) is preserved by construction: for every occurrence of an action a in a plan π for Π, there is a unique sequence of actions π'_a in $F^{\text{SDAC}}(\Pi)$ that is applicable in the state corresponding to s (with all fresh variables except for idle set to 0) iff a is applicable in s. This sequence π'_a is $\text{startAction}^a, \text{testPrecond}^a$ followed by appropriate $\text{copyToTape}^a_{i,1}$ and $\text{copyToTape}^a_{i,0}$ actions, startDTMSim^a, $\text{simDTMStep}^a_{a'}$ for each step of the DTM simulation, finishDTMSim^a, and appropriate $\text{evalCost}^a_{i,1}$ and $\text{evalCost}^a_{i,0}$ actions, concluded by an instance of finishAction^a.

Costs are preserved since the cost $C(a,s)$ of a in s is exactly reflected by the sum of costs of the applied $\text{evalCost}^a_{i,1}$ and $\text{evalCost}^a_{i,0}$ actions, and all other actions in $F^{\text{SDAC}}(\Pi)$ cost nothing. Also, the translation does not introduce any new plans that are not present in Π: Since the goal requires that idle is 1 and finishAction^a is the only action with this effect, a goal state can only be reached if a sequence of actions π'_a has been completely executed. Once an action sequence π'_a has been started, no further action sequence $\pi'_{a'}$ can be started, since idle is not set to 1 until the finishAction^a is executed, and thus the complete sequence π'_a has been executed. Finally, all actions of a sequence π'_a have a fixed order with mutually exclusive phases, which proves that the translation does not introduce any new plans.

All newly added propositional variables and actions not related to the simulation of the DTM used to compute the cost function are polynomially bounded by the size of the input task Π. Furthermore, the number of propositions and actions necessary to simulate the DTM is bounded by the space required of $\mathcal{T}$, i.e., $k_{\mathcal{T}}(|P|)$. Thus, since $\mathcal{C} \in$ FPSPACE, the compiled task is only polynomially larger than the input task. The resulting tasks are obviously CAC tasks, as $C'(s,a')$ does not depend on s. $\square$

Note that the SDAC translation only requires polynomial time. Finally, with Theorem 3, we prove that $\mathbf{f}^{\text{SDAC}}$ preserves plan length polynomially, provided a generic cost function in FP.

Theorem 3. *For tasks from a lifted domain* $\mathcal{L} = \langle \mathcal{D}, \mathcal{C}\rangle$ *with a generic cost function* $\mathcal{C}$ *in* FP*, the SDAC translation* $\mathbf{f}^{\text{SDAC}}$ *preserves plan length polynomially.*

Proof. Each action a of the original plan π for Π is replaced by a sequence of actions π'_a in the compiled task $\Pi' = F^{\text{SDAC}}(\Pi)$ as described in the proof of Theorem 2. Actions of a type that appear only once in π'_a, like startAction^a, are uncritical. Actions of the types $\text{copyToTape}^a_{i,1}$ or $\text{copyToTape}^a_{i,0}$, and $\text{evalCost}^a_{i,1}$ or $\text{evalCost}^a_{i,0}$ generally appear more than once, but their number is bounded by $k_{\mathcal{T}}(|P|)$, which, by assumption that $\mathcal{C} \in$ FP, is polynomial in size of Π.

The number of $\text{simDTMStep}^a_{a'}$ steps simulating the cost DTM of a is also bounded by a polynomial, as $\mathcal{C} \in$ FP. Since each action a of the original plan π for task Π is substituted by polynomially many actions π'_a, the entire length of the plan π' for Π' is also only polynomially longer than the original plan π. □

The compilability result for FP cost functions raises the question whether a similar translation also works for FNP cost functions. On the one hand, simulating an NTM instead of a DTM as part of a planning task is unproblematic, since action choices match the nondeterminism of an NTM, and unsuccessful runs of an NTM can be reflected in action sequences leading to dead-end states. On the other hand, FNP action cost functions, where the certificates to be verified are the cost values themselves, do not seem to be the appropriate complexity class to study when allowing nondeterminism. E. g., the cost function mapping an encoding of a propositional formula φ to 1 if φ is satisfiable, and to 0, otherwise, is *not* in FNP (unless $\mathsf{P} = \mathsf{NP}$), but rather in $\mathsf{FP}^{\mathsf{NP}}$. For oracle TMs, we are not aware of a simulation by a planning task where the plan length remains polynomially bounded. Instead of studying such oracle TMs further, we now focus our attention on two impossibility results involving FPSPACE and FP cost functions.

Impossibility Results

In the following, we prove two impossibility theorems, showing that it is impossible to compile SDAC with generic cost functions in FPSPACE away while preserving both task sizes and plan lengths polynomially, unless the polynomial hierarchy collapses at the third level (Theorem 4), and that it is impossible to compile SDAC with generic cost functions in FP away while preserving plan lengths linearly (Theorem 5).

Theorem 4. *There does not exist a compilation scheme compiling away SDAC with generic cost functions in* FPSPACE *that preserves plan length polynomially, unless the polynomial hierarchy collapses at the third level.*

Proof. Consider a lifted domain $\mathcal{L} = \langle \mathcal{D}, \mathcal{C} \rangle$ consisting of a family $\mathcal{D} = \{(P, A)_i^j\}_{i,j \in \mathbb{N}_0}$ of domain contexts such that P_i^j consists of i different propositional atoms, and j enumerates all such P_i^j with i different atoms. Action set $A_i^j = \{a\}$ consists of a single action a for all $i, j \in \mathbb{N}_0$. Action a can always be executed and produces a state satisfying the goal. The generic cost function $\mathcal{C}$ induces cost functions C that return 0 as the cost for executing a iff the initial state encodes a yes-instance of a PSPACE-complete problem (e.g., QBF), and 1 otherwise. Consider now the tasks Π_i^j, with $\mathbf{B} = 0$. Obviously, each planning task corresponds to an instance of the PSPACE-complete problem and we can use the generic cost function to decide the language.

Assuming now a compilation scheme as mentioned above would imply that we could produce tasks $F(\Pi_i^j)$ that are only polynomially larger than Π_i^j with successful plans that have length polynomially longer than the original plans. Since we had only one action in Π_i^j, successful plans of $F(\Pi_i^j)$ have a length that is polynomial in the size of Π_i^j, and therefore $F(\Pi_i^j)$. Deciding plan existence for planning tasks when only plans of polynomial length are allowed is, however, obviously a problem in NP. Making no assumption about the computational mechanism that generates the compiled tasks, it might be the case that for each domain size a different form of domain is generated, i.e., the compilation is essentially non-uniform. In other words, we could solve PSPACE-complete problems in NP/poly, i.e., $\mathsf{PSPACE} \subseteq \mathsf{NP/poly}$.[3] By a result by Yap (1983), this implies $\Sigma_p^3 = \text{co-}\Sigma_p^3$, i.e., a collapse of the polynomial hierarchy at the third level. □

Theorem 5. *There does not exist a compilation scheme compiling away SDAC with generic cost functions in* FP *that preserves plan length linearly.*

Proof. We use the same construction as above with the modification that instead of a PSPACE-complete problem we consider a generic cost function that decides PARITY, i.e., are there an even number of 1's. Assuming now a constant blowup of the plans by k (with perhaps an additive constant c), compiling Π_i^j to $F(\Pi_i^j)$, we have to consider only plans of length $k + c$. Since the domains can grow polynomially, there can only be $p(||\Pi_i^j||)$ different actions and therefore only $p(||\Pi_i^j||)^{k+c}$, i.e., polynomially many different plans. Each such plan could be translated into a circuit of constant depth. In other words, the problems, one can solve using planning domains with plans that have a constant number of actions in a plan are those in the class AC^0. It is, however, well-known that PARITY is not a member of AC^0 (Furst, Saxe, and Sipser 1984), hence, a compilation with the restrictions mentioned in the theorem is impossible. □

Discussion

Contrasting the result that preserving task sizes polynomially and plan lengths linearly at the same time is impossible (Theorem 5) with existing translations of state-dependent action costs, we can observe that the worst-case exponential task-size increase of the combinatorial translation (Geißer, Keller, and Mattmüller 2015) is unavoidable, as this translation preserves plan lengths exactly. This reasoning does not apply to the EVMDD-based translation (Geißer, Keller, and Mattmüller 2015), though, which has a low polynomial plan-length increase ($||\pi'|| \leq (|P| + 2) \cdot ||\pi||$).

Our results shed some additional light on the advantages and disadvantages of different ways of dealing with state-dependent action costs in planning models and algorithms. Those options include: (a) Do not compile state-dependent action cost away at all, but rather keep them in the model and support them natively in algorithms which avoids the overhead introduced by compilation. Representing cost functions declaratively, in closed form, is possible for functions such as polynomials. One successful example is the work on symbolic search with EVMDDs (Speck, Geißer, and Mattmüller 2018a,b). For more complicated cost functions

[3]Recall that NP/poly is the non-uniform analogue of NP, i. e., the class of problems solvable by an NTM with access to a polynomial-bounded advice function that provides an advice string to the NTM that may depend on the length of the input, but not on the input itself.

(e. g., the motion cost of a robot provided by an external motion planner), representing them using semantic attachments or similar mechanisms (Dornhege et al. 2009; Gregory et al. 2012; Haslum et al. 2018) may be advisable, but that comes at the price of making them less accessible to parts of planning algorithms such as goal-distance heuristics. (b) Compile them away using the DTM-based approach. This preserves task sizes polynomially, but fails to preserve plan lengths. We doubt the practical usefulness of such a compilation, though, and view it as a purely theoretical construction. Its heuristic-friendliness is also an open question. (c) Compile them away with the combinatorial translation. This necessarily leads to an exponential growth in task sizes while preserving plan lengths exactly. It is heuristic-friendly by construction. (d) Compile them away with the EVMDD-based translation. The increase in planning task size depends on the amount of additive separability of the cost functions. Highly additively separable functions like linear combinations of state features only lead to a polynomial increase in planning task size, while in the worst case, that increase can be exponential. Plan lengths are guaranteed to be preserved polynomially. This translation is often heuristic-friendly (Geißer 2018).

Notice that hybrid forms are also possible, like native handling of state-dependent cost in the search algorithm combined with a heuristic computation operating on a translation (Geißer 2018).

Conclusion

We established a comprehensive classification of compilability and non-compilability of state-dependent action costs for combinations of action cost functions and possible cost measures for the compilations. Generic cost functions with a computational complexity beyond FPSPACE were not considered, since planning with constant action costs is in FPSPACE itself, and hence planning complexity would be dominated by that of evaluating cost functions. The results for the case in which the planning task sizes ought to be preserved polynomially are summarized in Table 1.

Acknowledgments

David Speck was supported by the German Research Foundation (DFG) as part of the project EPSDAC (MA 7790/1-1). We thank the anonymous reviewers for their comments and suggestions.

References

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence* 69(1–2): 165–204.

Ciardo, G.; and Siminiceanu, R. 2002. Using Edge-Valued Decision Diagrams for Symbolic Generation of Shortest Paths. In Aagaard, M.; and O'Leary, J. W., eds., *Proceedings of the Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD 2002)*, volume 2517 of *Lecture Notes in Computer Science*, 256–273. Springer-Verlag.

Corraya, S.; Geißer, F.; Speck, D.; and Mattmüller, R. 2019. An Empirical Study of the Usefulness of State-Dependent Action Costs in Planning. In Benzmüller, C.; and Stuckenschmidt, H., eds., *Proceedings of the 42nd Annual German Conference on Artificial Intelligence (KI 2019)*, Lecture Notes in Artificial Intelligence, 123–130. Springer-Verlag.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic Attachments for Domain-Independent Planning Systems. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 114–121. AAAI Press.

Furst, M. L.; Saxe, J. B.; and Sipser, M. 1984. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory* 17(1): 13–27.

Geißer, F. 2018. *On Planning with State-dependent Action Costs*. Ph.D. thesis, University of Freiburg.

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete Relaxations for Planning with State-Dependent Action Costs. In Yang, Q.; and Wooldridge, M., eds., *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1573–1579. AAAI Press.

Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for Planning with State-Dependent Action Costs. In Coles, A.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, 140–148. AAAI Press.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning Modulo Theories: Extending the Planning Paradigm. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 65–73. AAAI Press.

Haslum, P.; Ivankovic, F.; Ramírez, M.; Gordon, D.; Thiébaux, S.; Shivashankar, V.; and Nau, D. S. 2018. Extending Classical Planning with State Constraints: Heuristics and Search for Optimal Planning. *Journal of Artificial Intelligence Research* 62: 373–431.

Ivankovic, F.; Gordon, D.; and Haslum, P. 2019. Planning with Global State Constraints and State-Dependent Action Costs. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 232–236. AAAI Press.

Nebel, B. 2000. On the Compilability and Expressive Power of Propositional Planning Formalisms. *Journal of Artificial Intelligence Research* 12: 271–315.

Savitch, W. J. 1970. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences* 4: 177–192.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018a. Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M., eds., *Proceedings of the Twenty-Eighth International*

Conference on Automated Planning and Scheduling (ICAPS 2018), 250–258. AAAI Press.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018b. SYMPLE: Symbolic Planning based on EVMDDs. In *Ninth International Planning Competition (IPC-9): planner abstracts*, 91–94.

Yap, C. 1983. Some Consequences of Non-Uniform Conditions on Uniform Classes. *Theoretical Computer Science* 26: 287–300.