Journal of Artificial Intelligence Research 82 (2025) 1349–1405 Submitted 07/2024; published 03/2025

Symbolic Search for Cost-Optimal Planning with Expressive Model Extensions

David Speck DAVIDJAKOB.SPECK@UNIBAS.CH
University of Basel, Switzerland

Jendrik Seipp JENDRIK.SEIPP@LIU.SE
Linköping University, Sweden

Álvaro Torralba ALTO@CS.AAU.DK
Aalborg University, Denmark

Abstract

In classical planning, the task is to derive a sequence of deterministic actions that changes the current fully-observable world state into one that satisfies a set of goal criteria. Algorithms for classical planning are domain-independent, i.e., they are not limited to a particular application and instead can be used to solve different types of reasoning problems. The main language for modeling such problems is the Planning Domain Definition Language (PDDL). Even though it provides many language features for expressing a wide range of planning tasks, most of today's classical planners, especially optimal ones, support only a small subset of its features. The most widely supported fragment is lifted STRIPS plus types and action costs. While this fragment suffices to model some interesting planning tasks, using it to model more realistic problems often incurs a much higher modeling effort. Even if modeling is possible at all, solving the resulting tasks is often infeasible in practice, as the required encoding size increases exponentially.

To address these issues, we show how to support more expressive modeling languages natively in optimal classical planning algorithms. Specifically, we focus on symbolic search, a state-of-the-art search algorithm that operates on sets of world states. We show how to extend symbolic search to support classical planning with conditional effects, axioms, and state-dependent action costs. All of these modeling features are *expressive* in the sense that compiling them away incurs a significant blow-up, so is it often necessary to support them natively. Except for blind (non-symbolic) search, our new symbolic search is the first optimal classical planning algorithm that supports these three modeling extensions in combination, and it even compares favorably to other state-of-the-art approaches that only support a subset of the extensions.

1. Introduction

Automated planning is the science of designing algorithms that can automatically derive behaviors to achieve goals. The generation of such strategies, or *thinking before acting*, is one of the original areas in the field of artificial intelligence. Informally, a classical planning problem is the task of finding a sequence of deterministic actions that allows an intelligent agent to achieve a set of goals from a given fully-observable initial state. Since planning is not restricted to a specific application, it was originally called general problem-solving (Newell & Simon, 1963; Helmert, 2008) and can be used for different types of reasoning problems, such as elevator control (Koehler & Schuster, 2000), greenhouse logistics (Helmert & Lasinger, 2010), natural language generation (Koller & Hoffmann,

2010), algorithm discovery (Speck, Höft, Gnad, & Seipp, 2023), robot control (Nilsson, 1984; Speck, Dornhege, & Burgard, 2017; Karpas & Magazzeni, 2020), network penetration testing (Speicher, Steinmetz, Backes, Hoffmann, & Künnemann, 2018; Torralba, Speicher, Künnemann, Steinmetz, & Hoffmann, 2021), wildfire fighting (Yu, Han, & Ma, 2014), and model checking (McMillan, 1993; Edelkamp, 2003b).

Most of today's optimal classical planning algorithms only support a basic subset of the features of the Planning Domain Definition Language (PDDL) (McDermott, Ghallab, Howe, Knoblock, Ram, Veloso, Weld, & Wilkins, 1998), namely lifted STRIPS plus types and action costs. For many real-world problems, however, this subset is not expressive enough to model the problem concisely. To model and solve such problems, it is often necessary to natively support more expressive language features.

In this article, we consider three model extensions of the *basic* planning model (Figure 1): conditional effects, derived predicates with axioms, and state-dependent action costs. All of these extensions allow us to capture different aspects of classical planning tasks while retaining the core of the classical planning formalism: single-agent planning problems in a fully observable, deterministic, static and discrete environment (Russell & Norvig, 2003). Two of these extensions, conditional effects and derived predicates, are already part of PDDL 2.2. Unfortunately, many classical planners do not support any of these expressive extensions. This is the case since many planners are based on heuristic search, and it is very challenging to design informative and fast-to-compute heuristics (goal-distance estimators) that take into account additional problem properties. This is especially true for cost-optimal planners, which additionally require that a heuristic is admissible, i.e., that the heuristic never overestimates the true cost of reaching a goal state. Therefore, even though the finite-domain representation (FDR) formalism used by the widely adopted Fast Downward Planning System (Helmert, 2006) considers conditional effects and derived predicates, most of the advanced techniques within Fast Downward or other derivative planners have to be disabled on problems that make use of these features.

However, it is well known that the three extensions that we consider here are crucial for the efficient and compact modeling of many real-world problems. Conditional effects provide a natural and compact way to model actions that have different outcomes depending on the context, i.e., the current state of the world (Nebel, 2000). Extending the state description to include derived variables allows aspects of a planning problem that are not directly affected by the actions but are derived from the values of other variables to be modeled concisely using a set of logical axioms (Thiébaux, Hoffmann, & Nebel, 2005). Thus, using derived predicates, it is possible to encode complex conditions (e.g., arbitrary Boolean formulas) to model the action's preconditions and/or goal. Finally, while in probabilistic planning a concise encoding of state-dependent action costs or rewards in form of Markov decision processes has long been standard, in classical planning compiling away state-dependent action costs incurs an exponential blow-up (Geißer, 2018). Furthermore, the three extensions are often required together. For example, actions with many conditional effects may require state-dependent action costs in order to make the cost dependent on which effects have been triggered.

Given the known complexity and compilability results for these model extensions (Nebel, 2000; Thiébaux et al., 2005; Speck, Borukhson, Mattmüller, & Nebel, 2021), it becomes evident that for many real-world problems it is desirable and even necessary to support

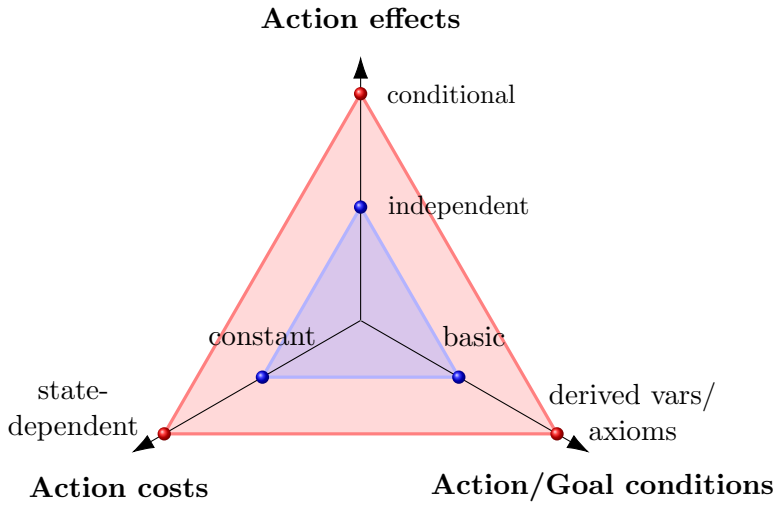


Figure 1: Overview of the dimensions in which we extend the basic planning formalism. The blue color represents the basic formalism, while the red color shows the formalism with our three expressive extensions: conditional effects, derived predicates with axioms, and state-dependent action costs. In this article, we show how symbolic search can support the planning formalism with these extensions.

these features natively. Symbolic search provides a suitable basis for such a native support, since state-of-the-art symbolic search methods do not require heuristics (Torralba, Alcázar, Kissmann, & Edelkamp, 2017; Speck, Geißer, & Mattmüller, 2020). In fact, symbolic *blind* search is very competitive and often complementary to other optimal classical planning techniques. For example, it is a key component of many modern planning systems that won awards such as the optimal track of the International Planning Competition in 2014 (Torralba, Alcázar, Borrajo, Kissmann, & Edelkamp, 2014), 2018 (Katz, Sohrabi, Samulowitz, & Sievers, 2018; Franco, Lelis, & Barley, 2018), and 2023 (Drexler, Gnad, Höft, Seipp, Speck, & Ståhlberg, 2023), and the Combinatorial Reconfiguration Competition in 2022 and 2023 (Christen, Eriksson, Katz, Muise, Petrov, Pommerening, Seipp, Sievers, & Speck, 2023). Since symbolic search does not necessarily require heuristics to be efficient, search efficiency does not suffer from the lack of efficient and informative heuristics for the more expressive planning formalisms. This is the main reason why, in contrast to most state-of-the-art heuristic search planners, it is feasible to support the discussed model extensions in modern symbolic blind search planners.

In this article, we provide a comprehensive overview and describe the key concepts of symbolic search for classical planning with the three expressive extensions (see Figure 1). More specifically, we describe theoretically and analyze empirically how symbolic search can support expressive model extensions with different symbolic data structures such as Binary Decision Diagrams (Bryant, 1986) or Edge-Valued Multi-Valued Decision Diagrams (Ciardo & Siminiceanu, 2002) in a unified framework. Based on this, we show that it is possible to support all model extensions simultaneously, resulting in optimal planning algorithms

that support conditional effects, derived predicates with axioms, and state-dependent action costs. Finally, our empirical evaluations demonstrate that the presented symbolic search algorithms complement and frequently show superior performance compared to other planning approaches from the literature. This holds across various planning domains, for each of the model extensions individually and in combination.

The outline of this article is as follows. We first present the necessary background for classical planning, decision diagrams, and symbolic search (Section 2), and the experimental setup we follow in all our empirical analyses (Section 3). We then consider planning with the following three expressive extensions in isolation: conditional effects (Section 4), axioms (Section 5), and state-dependent action costs (Section 6), and finally with all of them in combination (Section 7). In each of these sections, we formalize planning with the expressive extension, show how symbolic search can support the extension, and provide an empirical evaluation. Finally, we discuss future work and conclude the paper.

Preliminary versions of Section 5 and Section 6 were first presented by Speck, Geißer, Mattmüller, and Torralba (2019) and Speck, Geißer, and Mattmüller (2018a), respectively. Some underlying ideas of Section 4 can be found in the planner abstract by Kissmann, Edelkamp, and Hoffmann (2014) and in the work by Torralba et al. (2017). This article differs from the aforementioned works by providing support for the considered model extensions in a unified framework for symbolic search, including a comprehensive description that considers not only a single type of decision diagrams, but multiple types. In addition, we present extensive new experimental results that evaluate symbolic search in a unified framework with different decision diagrams and search directions (forward, backward, and bidirectional search), comparing its performance to other state-of-the-art techniques. For each of the model extensions and their combinations, we have collected relevant benchmarks and suitable baseline planners. To foster more research on model extensions, we make the benchmarks, the baseline planners and our code available online (Speck, Seipp, & Torralba, 2024). Finally, in Section 7 we provide new theoretical and empirical results showing that symbolic search can efficiently support all three considered model extensions simultaneously.

2. Background

In this section, we formally define classical planning and introduce one approach for it, symbolic search. Finally, we introduce different types of decision diagrams that are used as the underlying data structure in symbolic search.

2.1 Classical Planning Formalism

We consider the SAS$^+$ formalism for describing classical planning tasks (Bäckström & Nebel, 1995).

Definition 1 (SAS$^+$ Planning Task). A *SAS$^+$ planning task* is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{V}$ is a finite set of state variables v, each associated with a finite domain D_v. A *fact* is a pair $\langle v, w \rangle$, where $v \in \mathcal{V}$ and $w \in D_v$. A *partial state* s over $\mathcal{V}(s) \subseteq \mathcal{V}$ is a function such that $s(v) \in D_v$ for all $v \in \mathcal{V}(s)$. If s assigns a value to each variable $v \in \mathcal{V}$, s is called a *state*. We often treat (partial) states as sets of facts, i.e., $s = \{\langle v, s(v) \rangle \mid v \in \mathcal{V}(s)\}$.

With $\mathcal{S}$ we refer to the set of all states that can be defined over $\mathcal{V}$. $\mathcal{O}$ is a finite set of *operators/actions*, where each operator is a pair $o = \langle pre_o, eff_o \rangle$ of partial states, called *preconditions* and *effects*. An operator $o \in \mathcal{O}$ is applicable in a state s iff pre_o is satisfied in s, i.e., $pre_o \subseteq s$. Applying operator o to state s yields state $s' = s[\![o]\!]$, where $s'(v) = eff_o(v)$ for all variables $v \in \mathcal{V}$ for which eff_o is defined and $s'(v) = s(v)$ otherwise. Furthermore, Π consists of the cost function $\mathcal{C} : \mathcal{O} \to \mathbb{N}_0$ that describes the cost of applying an operator.[1] With $range(\mathcal{C}) = \{\mathcal{C}(o) \mid o \in \mathcal{O}\}$, we refer to the set of all possible cost values of the operators. Finally, $\mathcal{I} \in \mathcal{S}$ is the *initial state* of Π and the partial state $\mathcal{G}$ (*goal condition*) defines the set of goal states $S_\star = \{s \in \mathcal{S} \mid \mathcal{G} \subseteq s\}$.

The objective of classical planning is to find plans, which are sequences of applicable operators leading from the initial state to a goal state, or to prove that no such sequence exists.

Definition 2 (Plan). A *plan* $\pi = \langle o_0, \ldots, o_{n-1} \rangle$ for planning task Π is a sequence of applicable operators that generates a sequence of states $s_0, \ldots, s_n$, where $s_0 = \mathcal{I}$, $s_n \in S_\star$ is a goal state, $s_{i+1} = s_i[\![o_i]\!]$ for all $i = 0, \ldots, n-1$. The *cost* of a plan is the sum of its operator costs, i.e., $cost(\pi) = \sum_{i=0}^{n-1} \mathcal{C}(o_i)$. A plan is *optimal* if there is no cheaper plan.

The search for an optimal plan is called *cost-optimal planning*, or *optimal planning* for short, and is the focus of this article. Example 1 describes a Mars rover planning task that can easily be encoded using the SAS$^+$ formalism from Definition 1. We will use this example throughout this article to motivate and explain expressive extensions to the SAS$^+$ formalism.

Example 1. Consider a Mars rover similar to Perseverance[2] that is designed to perform autonomous tasks. Such a scenario is illustrated in Figure 2. The dynamics of this example are as follows. The rover can navigate between adjacent cells if they are free (impassable cells are highlighted in red). Navigating the rover between cells has no cost, i.e., a cost of 0. There are interesting rocks at certain locations, and the rover can collect a sample of those rocks at a cost of 1 when it is at that location. We consider the particular planning task shown in Figure 2, where the rover is initially located at (7,3), the actual landing site of Perseverance. The goal is to collect rock samples at (5,1) and (7,1) and bring the samples to (0,5), a location known as "Three Forks" from which new missions can be launched. The cost of gathering a rock sample is 1. An optimal plan for this task is $\pi = \langle$*navigate*-7-2, *navigate*-7-1, *sample-rock*-7-1, *navigate*-7-0, ..., *navigate*-5-1, *sample-rock*-5-1, *navigate*-5-0, ..., *navigate*-0-5$\rangle$ with a cost of $\mathcal{C}(\pi) = 2$, since the *navigate* actions cost 0 and there are two *sample-rock* actions, each incurring cost 1.

For the remainder of this paper, we assume binary variables, unless stated otherwise, to simplify the presentation. We assume that $D_v = \{0, 1\}$, and write v as a shorthand for $\langle v, 1 \rangle$ and $\neg v$ for $\langle v, 0 \rangle$. We emphasize that the synthesis of finite domain variables using mutexes (Helmert, 2009) combined with a binary encoding is critical for the efficiency of symbolic search with BDDs (Edelkamp & Helmert, 1999, 2001). Such a binary encoding is possible

1. In this article, we stick to the common practice of considering operator cost functions with the natural numbers $\mathbb{N}_0$ as the codomain, which allow efficient cost bucketing within symbolic search algorithms.
2. `https://mars.nasa.gov/mars2020/mission/overview/` (Accessed: 2023-09-21)

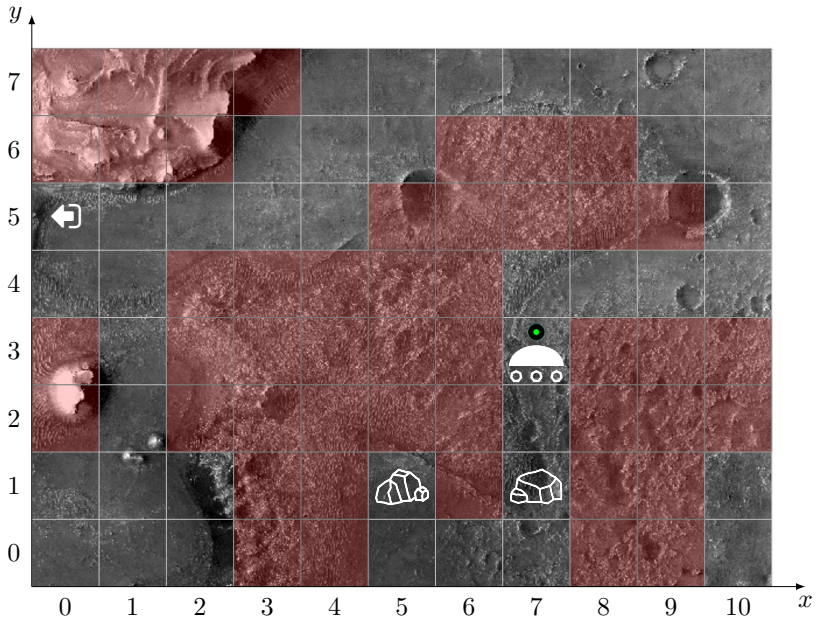


Figure 2: Visualization of the running example, a Mars rover planning task. The original image is from NASA/JPL-Caltech/University of Arizona[3] and shows the Jezero crater, where the green dot indicates the actual landing site of the Perseverance rover. To illustrate the task, we added the rover, grid lines, red coloring (impassable cells for the rover), rocks (goal: collect sample), and an arrow to indicate where the rover should end its journey.

for all SAS$^+$ planning tasks because each finite domain variable $v \in \mathcal{V}$ can be represented by $\lceil \log_2 |D_v| \rceil$ binary variables. All the theory presented, and our implementations support finite-domain variables. The restriction to binary variables in the following is only for ease of understanding.

2.2 Decision Diagrams

Decision diagrams are data structures that can be used to represent relevant functions for solving a planning task. For example, we can represent a *set* of states $S \subseteq \mathcal{S}$ by its *characteristic function* χ_S, which is a Boolean function $\chi_S : \mathcal{S} \to \{0, 1\}$ that decides whether a given state belongs to S or not. More precisely, $\chi_S(s) = 1$ if $s \in S$ and $\chi_S(s) = 0$ otherwise.

In symbolic search, the most prominent way to represent (characteristic) functions is to use decision diagrams such as binary decision diagrams (BDDs) (Bryant, 1985), algebraic decision diagrams (ADDs) (Bahar, Frohm, Gaona, Hachtel, Macii, Pardo, & Somenzi, 1997) or edge-valued binary decision diagrams (EVBDDs) (Lai, Pedram, & Vrudhula, 1996). All of these data structures offer a compromise between conciseness of representation and efficiency

3. `https://mars.nasa.gov/resources/25621/perseverances-landing-spot-in-jezero-crater/` (Accessed: 2023-09-21)

of manipulation (Drechsler & Becker, 1998a). Their main idea is to break down a function f into subfunctions, so that f can be reassembled from them.

Definition 3 (Binary Decision Diagram). Let X be a set of binary variables and let χ be a Boolean function over X. A *binary decision diagram* (BDD) B_χ is a directed acyclic graph with a single root node and two terminal nodes: the 0-sink and the 1-sink. Each inner node corresponds to a binary variable $x \in X$ and has two successors, where the *low edge* represents that variable x is false, while the *high edge* represents that variable x is true. The represented Boolean function χ is evaluated by traversing the BDD according to a given assignment.

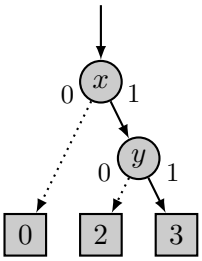
As we explain in Section 2.3, we consider BDDs with $X = \mathcal{V}$ to represent a set of states, and with $X = \mathcal{V} \cup \mathcal{V}'$ to represent sets of operators, where $\mathcal{V}' = \{v' \mid v \in \mathcal{V}\}$ denotes the set of primed versions of the state variables $\mathcal{V}$.

While BDDs are commonly used to represent state sets, ADDs and EVBDDs have been successfully used to represent numerical functions, e.g., mappings from states to a numerical value $f : \mathcal{S} \to \mathbb{Q} \cup \{\infty\}$ in symbolic planning (Hansen, Zhou, & Feng, 2002; Torralba, Linares López, & Borrajo, 2013; Speck, Geißer, & Mattmüller, 2018b). An *algebraic decision diagram* (ADD) A_f is similar to a BDD, but it has an arbitrary number of terminal nodes with different discrete values, including real numbers. Edge-valued multi-valued decision diagrams (EVMDDs) are rooted directed acyclic graphs that generalize their binary counterparts (EVBDDs), by allowing variables to have multiple values (Ciardo & Siminiceanu, 2002). Due to their generality, EVMDDs are more common in planning than EVBDDs (Geißer, Keller, & Mattmüller, 2015, 2016; Mattmüller, Geißer, Wright, & Nebel, 2018; Speck et al., 2018a).

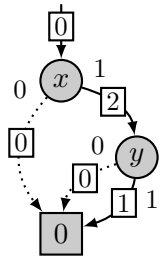
Definition 4 (Edge-Valued Multi-valued Decision Diagram). Let $\mathcal{V}$ be a set of multi-valued variables and let f be a function over $\mathcal{V}$. An *edge-valued multi-valued decision diagram* (EVMDD) $\mathcal{E}_f$ is a weighted directed acyclic graph with a single dangling incoming edge with weight $w \in \mathbb{Q} \cup \{\infty\}$ to a root node and a single terminal node denoted by 0. Each inner node corresponds to a multi-valued variable $v \in \mathcal{V}$ and has, for each domain value $i \in D_v$, an outgoing edge to a successor node n_i with edge weight $w_i \in \mathbb{Q} \cup \{\infty\}$ such that $\min_{i \in D_v} w_i = 0$. The represented function f is evaluated by traversing the graph according to the variable assignment, summing the edge weights to yield the function value.

Decision diagrams are typically considered in a reduced and ordered form and can represent exponentially many states requiring only polynomial space. A decision diagram is called *ordered* if variables appear in the same order on all paths from the root to a sink. A decision diagram is called *reduced* if isomorphic subgraphs are merged and any node is eliminated whose two children are identical. For fixed variable orders, reduced and ordered decision diagrams are unique (Bryant, 1986; Bahar et al., 1997; Lai et al., 1996). Note that for EVBDDs and EVMDDs the corresponding edge values must be taken into account. From now on, we consider all decision diagrams to be reduced and ordered for a fixed variable order.

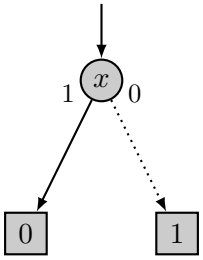
Example 2. Figures 3a and 3b show the ADD A_f and EVMDD E_f representing the numeric function $f = 2x + xy$. The function f can also be represented as multiple BDDs by



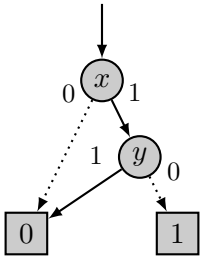
(a) ADD A_f representing the numeric function $f = 2x + xy$ with size $|A_f| = 5$.



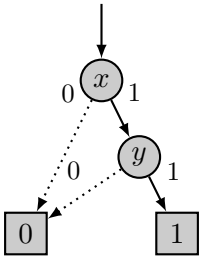
(b) EVMDD E_f representing the numeric function $f = 2x + xy$ with size $|E_f| = 3$.



(c) BDD $B_{f=0}$ representing the characteristic function $\chi_S = \neg x$ with size $|B_{f=0}| = 3$.



(d) BDD $B_{f=2}$ representing the characteristic function $\chi_S = x \wedge \neg y$ with size $|B_{f=2}| = 4$.



(e) BDD $B_{f=3}$ representing the characteristic function $\chi_S = x \wedge y$ with size $|B_{f=3}| = 4$.

Figure 3: Visualization of different decision diagrams used in Example 2.

disassembling the ADD A_f into three different BDDs, one for each terminal node. Figures 3c to 3e depict the BDDs $B_{f=z}$ representing all states for which the evaluation of function $f = 2x + xy$ is z. The variable order for all decision diagrams is $x \succ y$, i.e., x appears before y on each path.

The *size* $|D|$ of a decision diagram D is the number of nodes in D. The size of a decision diagram strongly depends on the variable order, so that a good order can lead to an exponentially more compact decision diagram (Edelkamp & Kissmann, 2011). For some functions the size of the corresponding decision diagram is exponential, independent of the underlying variable order (Bryant, 1986; Edelkamp & Kissmann, 2011).

Comparing the different types of decision diagrams, an EVBDD can be exponentially more compact than an ADD (Siminiceanu & Roux, 2010) representing additively separable functions such as $f : \{0,1\}^{n+1} \to \{0, \ldots, 2^{n+1} - 1\}$ with $f(x_0, \ldots, x_n) = \sum_{i=0}^{n} 2^i x_i$. Moreover, an ADD can be efficiently disassembled into multiple BDDs, one for each terminal node, in polynomial time and memory with respect to the ADD size (Torralba, 2015).

In practice, the main advantage of using BDDs over ADDs (and EVMDDs) is that decision diagram libraries such as CUDD (Somenzi, 2015) use techniques like complement edges to store BDDs more compactly (Brace, Rudell, & Bryant, 1990) and allow for more efficient operations (Burch, Clarke, Long, McMillan, & Dill, 1994). However, the use of ADDs or EVMDDs as data structures in symbolic search makes it possible to simultaneously represent multiple sets of states associated to different costs. As shown in Figure 2, this can lead to a more concise symbolic representation.

In the text below, when we refer to state sets, characteristic functions and numerical functions, we assume that they are represented as one of the corresponding decision diagrams and all logical or numerical operations are realized with the efficient and appropriate decision diagram-based operations using the APPLY algorithm (Bryant, 1986; Bahar et al., 1997; Lai et al., 1996). In this article, we consider symbolic search with BDDs and EVMDDs. While alternative representations exist in both cases (e.g., ZDDs (Minato, 1993), or Affine ADDs (Sanner & McAllester, 2005)), BDDs and EVMDDs have shown good performance for symbolic search in planning.

2.3 Symbolic Search with BDDs

Symbolic search algorithms, originally developed in the field of model checking (McMillan, 1993), are similar to their explicit counterparts. However, they differ in that symbolic search generates and expands entire *sets* of states, as opposed to individual states. To enable this approach, a SAS$^+$ planning task (Definition 1) must be represented in a symbolic way (Edelkamp & Helmert, 2000, 2001; Edelkamp & Kissmann, 2009; Torralba, 2015; Torralba et al., 2017), which is outlined below using BDDs as the underlying data structure.

The BDDs representing the characteristic function of the initial state and the states satisfying the goal condition of a SAS$^+$ task can be constructed in linear time in the number of variables $|\mathcal{V}|$, since they are simply a conjunction of facts. Note that this is the case even for tasks with an exponential number of goal states.

Similarly, single operators $o \in \mathcal{O}$ or sets of operators $O \subseteq \mathcal{O}$ are represented as so-called *transition relations*, which are sets of state pairs, namely predecessor and successor states. The characteristic function of a transition relation T_O is a function $\chi_{T_O} : \mathcal{S} \times \mathcal{S} \to \{0, 1\}$ that maps all pairs of states $\langle s, s' \rangle$ to 1 iff there exists $o \in O$ such that $pre_o \subseteq s$ and $s' = s[\![o]\!]$. In practice, we use two sets of variables $\mathcal{V}$ and $\mathcal{V}'$, one for the current states s, which we describe by a set of unprimed variables $\mathcal{V} = \{v_0, \ldots, v_{|\mathcal{V}|-1}\}$, and another for the successor states s', which we describe by a set of primed variables $\mathcal{V}' = \{v'_0, \ldots, v'_{|\mathcal{V}|-1}\}$. For a formula or a partial state ϕ, we write $\phi[\mathcal{V}]$ when referring to ϕ encoded using the unprimed variables $\mathcal{V}$. Similarly, with $\phi[\mathcal{V}']$ we refer to encoding ϕ using the primed variables $\mathcal{V}'$.

Definition 5 (Transition Relation). The transition relation T_o of an operator $o \in \mathcal{O}$ is defined as $T_o = pre_o[\mathcal{V}] \wedge eff_o[\mathcal{V}'] \wedge \bigwedge_{v \in \mathcal{V} \setminus \mathcal{V}(eff_o)}(v \iff v')$. Multiple operators $O \subseteq \mathcal{O}$ with the same cost can be represented by a single transition relation as $T_O = \bigvee_{o \in O} T_o$.

In Definition 5, the transition relation consists of three main parts. The first part encodes the preconditions over the unprimed variables, the second part encodes the effects over the primed variables, and the last part ensures the closed-world assumption by encoding so-called *frame axioms*, meaning that any variable that is not changed by the effect keeps its original value.

Example 3. Consider a set of two Boolean variables $\mathcal{V} = \{x, y\}$ and an operator $o = \langle \{x\}, \{\neg y\} \rangle$. The transition relation for o is $T_o = x \wedge \neg y' \wedge (x \iff x') = x \wedge x' \wedge \neg y'$. T_o can also be interpreted as the set of pair of states (s, s') such that o is applicable on s resulting on s'. In the example, this correspond to two pairs of states: $(\langle x = 1, y = 1 \rangle, \langle x' = 1, y' = 0 \rangle)$ and $(\langle x = 1, y = 0 \rangle, \langle x' = 1, y' = 0 \rangle)$.

Finally, given a set of states S and a transition relation T_O, the *image* operator computes the set of successor states S' of S via T_O, i.e., $S' = \{s' \in \mathcal{S} \mid \exists s \in S, o \in O \text{ s.t. } pre_o \subseteq s \text{ and } s' = s[\![o]\!]\}$. Analogously, given a set of states S' and a transition relation T_O, the *preimage* operator computes the set of predecessor states S' of S via T_O. The exact details of how these operations work are not important for following the ideas presented in this article. Note, however, that in practice, most BDD packages implement efficient image and preimage operations in the form of the so-called *relational product* (Burch et al., 1994). The key take-away is that the method above allows us to encode a SAS$^+$ task as BDDs.

Definition 6 (Symbolic Task Representation with BDDs). The symbolic task representation with BDDs of a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$ is a tuple $\langle \mathcal{T}, \chi_{\mathcal{I}}, \chi_{\mathcal{G}} \rangle$ where $\chi_{\mathcal{I}}[\mathcal{V}]$ is a BDD representing the initial state, $\chi_{\mathcal{G}}[\mathcal{V}]$ is a BDD representing the set of goal states, and $\mathcal{T}$ is a set of transition relations with a transition relation $T_o[\mathcal{V}, \mathcal{V}'] \in \mathcal{T}$ for each operator $o \in \mathcal{O}$.

Even if the transition relation represents exponentially many state pairs, the size of the BDD representing the transition relation of a single SAS$^+$ operator is always linear in the number of variables (assuming that every pair of variables x, x' are adjacent in the variable ordering). This is the case because the precondition and effect are simple conjunctions of facts, whereas $(x \iff x')$ can be represented with a constant number of nodes if the variables are next to each other in the variable ordering. In the following, when we refer to the size of a transition relation, we mean the size of the decision diagram representing it.

Next, we describe algorithms that can find an optimal plan for tasks in this representation. Importantly, the transition relation, as well as the image and preimage, do not need any specific form. This means that we can model and handle any predecessor-successor relation, and we are not constrained by, for example, conjunctive preconditions in the form of partial states or context-free conjunctive effects. As long as the transition relation accurately encodes the predecessor-successor state relation of operators, the image (or preimage) operator computes the correct set of successor (or predecessor) states for a given set of states. Therefore, the key concept in supporting expressive extensions to the SAS$^+$ planning formalism (Definition 1) is to correctly encode the underlying predecessor-successor relation of operators with feature extensions, such as conditional effects or derived predicates.

Symbolic (blind) search is a symbolic version of uniform cost search, also known as Dijkstra's algorithm (Dijkstra, 1959), which can be performed in different search directions. *Symbolic forward (blind) search*, also known as progression, begins with the representation of the initial state $\chi_{\mathcal{I}}$. Then, it iteratively computes the image using transition relations that represent the operators of the planning task at hand until it finds a set of states S whose intersection with the goal $\chi_{\mathcal{G}}$ is non-empty, i.e., $\chi_S \wedge \chi_{\mathcal{G}} \neq \bot$. The *open* and *closed lists* are represented as state sets partitioned into buckets with identical g-values, where the *g-value* is the cost required to reach the set of states in a bucket. During the search, the closed list is used to track and prune states that have already been expanded. Algorithm 1 shows pseudo-code for symbolic forward search with non-zero operator costs, which is arguably the simplest symbolic search algorithm. For simplicity, we only present symbolic search algorithms (Algorithm 1 and later Algorithm 3) for non-zero operator costs. However, our theory generalizes to zero-cost operators, as described by Torralba, Speck et al. (2015, 2018a), and our implementations support them.

Algorithm 1: Symbolic BDD forward search with non-zero operator costs.

Data: Planning task $\langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$ and BDD representation $\langle \mathcal{T}, \chi_{\mathcal{I}}, \chi_{\mathcal{G}} \rangle$
Result: Optimal plan π

1 $T_{O_c} \leftarrow \bigvee_{T_o \in \mathcal{T}, \mathcal{C}(o)=c} T_o$ for all $c \in \mathit{range}(\mathcal{C})$
2 $\mathit{open}_0 \leftarrow \chi_{\mathcal{I}}$
3 $\mathit{closed}_g \leftarrow \bot$ for all $g \in \mathbb{N}_0$
4 **while** $\exists g : \mathit{open}_g \neq \bot$ **do**
5 $\quad g_{min} \leftarrow \mathit{min}\{g \mid \mathit{open}_g \neq \bot\}$
6 $\quad \mathit{closed}_{g_{min}} \leftarrow \mathit{open}_{g_{min}} \wedge \bigwedge_{g=0}^{g<g_{min}} \neg \mathit{closed}_g$
7 $\quad \mathit{open}_{g_{min}} \leftarrow \bot$
8 $\quad$ **if** $(\mathit{closed}_{g_{min}} \wedge \chi_{\mathcal{G}}) \neq \bot$ **then**
9 $\quad\quad$ **return** *reconstruct-plan*$(\Pi, \mathit{closed}_{g_{min}} \wedge \chi_{\mathcal{G}}, \langle \mathit{closed}_0, \ldots, \mathit{closed}_{g_{min}} \rangle)$
10 $\quad$ **forall** $c \in \mathit{range}(\mathcal{C})$ **do**
11 $\quad\quad \mathit{open}_{g_{min}+c} \leftarrow \mathit{open}_{g_{min}+c} \vee \mathit{image}(\mathit{closed}_{g_{min}}, T_{O_c})$
12 **return** task is unsolvable

The first line of Algorithm 1 aggregates all the transition relations of operators with the same cost into a single transition relation. This is an optional step but recommended, as it significantly speeds up the image computation. However, it also may increase the memory required to represent the transition relation. As the size of the aggregated transition relation is worst-case exponential in the number of operators with the same cost, a time and memory limit can be used to ensure the step terminates successfully (Torralba et al., 2017). The algorithm continues by initializing the open list bucket open_0 with a g-value of 0, which represents the singleton set containing the initial state, and the closed list, which is empty at the beginning. Until all open list buckets open_g are empty (line 4), the algorithm iteratively chooses the cheapest-to-reach, non-empty bucket $\mathit{open}_{g_{min}}$ (line 5) and expands it. Expanding $\mathit{open}_{g_{min}}$ involves removing all previously expanded states with lower cost and then marking the remaining set of states as closed (line 6). If a goal state is reached (line 8), a plan is reconstructed (line 9), the details of which are explained below. As the final step of a bucket expansion, the algorithm generates the set of successor states by iterating over the transition relations T_{O_c} representing all operators O_c with cost c, computing the image with respect to $\mathit{closed}_{g_{min}}$ and T_{O_c}, and then adding the new states to the corresponding open list buckets $\mathit{open}_{g_{min}+c}$ (lines 10 and 11). If all open list buckets become empty and no goal state was reached, the task is unsolvable because all reachable states have been considered (line 12).

Symbolic backward (blind) search, also known as regression, starts with the goal states, and applies the preimage operation until the initial state is found. In *symbolic bidirectional (blind) search*, both forward and backward symbolic search are performed simultaneously, maintaining two separate open and closed lists (Torralba et al., 2017). A search step consists either of a backward or a forward search step (and modifies the respective open and closed lists). A goal path is found once the current search direction expands a symbolic state set that shares an explicit state with a symbolic state contained in the closed list of the opposite

search direction. Note that in bidirectional search, the search cannot immediately stop once a state is found in both directions; it must first be proven to be on an optimal path. In symbolic bidirectional blind search, optimality is ensured by expanding a state that has already been closed in the opposite direction. When considering bidirectional search with heuristics (which we do not consider in this paper), more sophisticated stopping conditions are required (Holte, 2010).

Finally, symbolic search needs a *plan reconstruction* procedure to obtain the final plan (Torralba, 2015). In explicit search, each search node keeps track of its parent node, making it easy to construct a plan when a goal state is found. In symbolic search, however, the parents are not directly known, but all parents are stored in the closed list with their reachability costs. Therefore, it is possible to perform a greedy search, which opposes the actual search direction, using the optimal path costs stored in the closed list as the perfect heuristic estimates. In symbolic forward search, the plan is reconstructed by a greedy backward search starting with an explicit goal state that was found by the symbolic search. Iteratively, the plan reconstruction procedure loops over all operators and selects an explicit predecessor state contained in the closed list. The procedure ends when the initial state is reached. Plan reconstruction in symbolic search can also be solved with a divide-and-conquer approach that avoids storing all closed lists with their reachability costs, which trades memory consumption for runtime (Jensen, Hansen, Richards, & Zhou, 2006).

Algorithm 2: Plan reconstruction for symbolic forward search.

```
   Data: Planning task Π = ⟨V, O, C, I, G⟩
   Data: Target states target
   Data: Closed list buckets ⟨closed_0, ..., closed_{g_min}⟩
   Result: Optimal plan π
 1 π ← ⟨⟩
 2 g ← g_min
 3 s ← select arbitrary state from target
 4 while s ≠ I do
 5     foreach o ∈ O do
 6         pred ← preimage(χ_s, T_{o})
 7         g_pred ← g − C(o)
 8         if pred ∧ closed_{g_pred} ≠ ⊥ then
 9             s ← select arbitrary state from pred
10             g ← g_pred
11             prepend o to π
12             break
13 return π
```

Algorithm 2 details the plan reconstruction for the symbolic forward algorithm described in Algorithm 1. Lines 1 to 3 initialize the plan π as an empty sequence, initialize the variable g with the remaining plan cost of g_{min}, and assign to s an arbitrary explicit goal state from the set of goal states *target*. The procedure continues to reconstruct the plan backwards until we reach the initial state (line 4). In line 5, we iterate over all operators, and in lines 6

and 7, we compute the set of predecessor states with respect to the selected operator and our current state, and calculate the remaining plan cost of those predecessors. In line 8, we check whether we have found a predecessor with such costs during search, i.e., whether there is such a predecessor in the respective closed list. If so, in lines 9 to 12 we select one of these predecessors from which we continue the plan reconstruction, update the remaining plan costs, prepend the determined operator to our plan, and continue the main loop of line 4. Finally, when we reach the initial state, we return the reconstructed plan π.

In backward symbolic search, the plan reconstruction procedure is similar, but it starts with the initial state and selects explicit successor states contained in the closed list. Note that a greedy search in combination with the perfect heuristic leads the search directly from a starting state to a target state, making the runtime of the plan reconstruction negligible with respect to the actual search. For bidirectional search, a greedy best-first search is performed twice, both opposing the actual search direction. More specifically, both plan reconstructions are initialized with an explicit state contained in the symbolic meeting point and one search is a regression to the initial state, while the other search is a progression to the goal states.

2.4 Symbolic Search with EVMDDs

Symbolic Search with EVMDDs operates similar as with BDDs. However, EVMDDs enable a richer representation of numerical functions, which can be used to represent mappings from states to numbers, $f : \mathcal{S} \to \mathbb{Q} \cup \{\infty\}$, and weighted transition relations that represent the cost of the cheapest operator that can be applied in s to get to s', $f : \mathcal{S} \times \mathcal{S}' \to \mathbb{Q} \cup \{\infty\}$.

So, while symbolic search with BDDs keeps separate sets of states per g-value in the open list, EVMDDs can represent the entire open list in a single decision diagram. An EVMDD $\mathcal{E}[\mathcal{V}]$ represents a set of states where each state in the set is assigned a finite cost and any state not in the set is assigned ∞. The example from Figure 3 can be interpreted as an open list with sets of states that are reachable from the initial state with a cost of 0, 2, and 3, respectively. When using an EVMDD, the single diagram in Figure 3b can represent all these sets of states even if they are at different distances from the initial state. Note that the idea of representing states with different assigned priority values within a decision diagram is also common practice in symbolic heuristic search (Reffel & Edelkamp, 1999; Hansen et al., 2002; Speck et al., 2018a).

Whenever we can construct a BDD to represent a set of states χ_S, this is equivalent to having an EVMDD that represents the function $s \mapsto 0$ if $s \in S$ and $s \mapsto \infty$ otherwise. Also, we can manipulate sets of states as with BDDs. For example, given two EVMDDs $\mathcal{E}_1$, $\mathcal{E}_2$, the *union-min* ($\overset{\min}{\vee}$) operation assigns each state s the value $\min(\mathcal{E}_1(s), \mathcal{E}_2(s))$, resulting in the union of both sets where each state s is assigned the minimum value of the two functions. Similarly, for the *intersection-max* ($\overset{\max}{\wedge}$) operation, each state s is assigned $\max(\mathcal{E}_1(s), \mathcal{E}_2(s))$, resulting in the intersection of both sets where each state is assigned the maximum value of the two functions. We also consider the negation and preserve-min operations for EVMDDs (Speck et al., 2018a). For arithmetic functions, there is no traditional complement; we define the *complement* of an EVMDD $\mathcal{E}$ as the function $\neg\mathcal{E}(s) = 0$ if $\mathcal{E}(s) = \infty$ and $\neg\mathcal{E}(s) = \infty$ otherwise. Note that this definition of complement is not self-inverse. The preserve-min operation extracts the least-cost states from an EVMDD. Given an EVMDD

$\mathcal{E}$, the *preserve-min* operation is defined as $\textit{preserve-min}(\mathcal{E})(s) = \mathcal{E}(s)$ if $\mathcal{E}(s) = \min_{s \in \mathbb{S}} \mathcal{E}(s)$ and $\textit{preserve-min}(\mathcal{E})(s) = \infty$ otherwise.

While symbolic search with BDDs keeps separate transition relations per operator cost, EVMDDs can represent the transition relation of all operators in a single decision diagram. A transition relation in EVMDDs $T_o^{\mathcal{E}}[\mathcal{V}, \mathcal{V}']$ corresponding to an operator $o \in \mathcal{O}$, is an EVMDD that represents the function over $(s, s') \mapsto \mathcal{C}(o)$ if $(s, s') \in T_o^{\mathcal{E}}$ and $(s, s') \mapsto \infty$ otherwise.

Definition 7 (Symbolic Task Representation with EVMDDs). The symbolic task representation with EVMDDs of a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$ is a tuple $\langle \mathcal{T}^{\mathcal{E}}, \mathcal{E}_{\mathcal{I}}, \mathcal{E}_{\mathcal{G}} \rangle$ where $\mathcal{E}_{\mathcal{I}}[\mathcal{V}]$ is an EVMDD representing the initial state, $\mathcal{E}_{\mathcal{G}}[\mathcal{V}]$ is an EVMDD representing the set of goal states, and $\mathcal{T}^{\mathcal{E}}$ is a set of transition relations with a transition relation $T_o^{\mathcal{E}}[\mathcal{V}, \mathcal{V}'] \in \mathcal{T}$ for each operator $o \in \mathcal{O}$.

Algorithm 3 shows how to perform blind search with EVMDDs, a simplified form of EVMDD-A* proposed by Speck et al. (2018a).

Algorithm 3: Symbolic EVMDD forward search with non-zero operator costs.

Data: Planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$ and EVMDD representation $\langle \mathcal{T}^{\mathcal{E}}, \mathcal{E}_{\mathcal{I}}, \mathcal{E}_{\mathcal{G}} \rangle$
Result: Optimal plan π

1 $T \leftarrow \bigvee^{\min}_{T_o^{\mathcal{E}} \in \mathcal{T}^{\mathcal{E}}} T_o^{\mathcal{E}}$
2 *open* $\leftarrow \mathcal{E}_{\mathcal{I}}$
3 *closed* $\leftarrow \mathcal{E}_{\infty}$
4 **while** *open* $\neq \infty$ **do**
5 $\quad \mathcal{E} \leftarrow \textit{preserve-min}(\textit{open})$
6 $\quad \textit{closed} \leftarrow \textit{closed} \overset{\min}{\vee} \mathcal{E}$
7 $\quad$ **if** $\mathcal{E} \overset{\max}{\wedge} \mathcal{E}_{\mathcal{G}} \neq \infty$ **then**
8 $\quad\quad$ **return** $\textit{reconstruct-plan}(\Pi, \mathcal{E} \overset{\max}{\wedge} \mathcal{E}_{\mathcal{G}}, \textit{closed})$
9 $\quad \textit{open} \leftarrow \textit{open} \overset{\min}{\vee} \textit{image}(\mathcal{E}, T)$
10 $\quad \textit{open} \leftarrow \textit{open} \overset{\max}{\wedge} \neg \textit{closed}$
11 **return** task is unsolvable

As mentioned above, we can construct a single transition relation that represents all operators (line 1). Similar to aggregating transition relations with the same cost using BDDs, representing multiple or all transition relations as monolithic EVMDDs can exceed memory and time resources. Therefore, it is common practice to impose time and memory limits to ensure the process terminates (Torralba et al., 2017; Speck et al., 2018a). Then, in lines 2 and 3, the open list is initialized with the EVMDD representing the initial state, and the closed list is initialized as the empty set. The algorithm continues as long as any state remains in the open list. In line 5, a set of states is extracted with the *preserve-min* operation, which keeps states with the lowest costs (minimal g-values) and maps all others to infinity. Subsequently, those states are inserted into *closed*. If any state of $\mathcal{E}$ is a goal state, the plan is reconstructed and returned. In contrast to Algorithm 1 which partitions the open and closed lists into buckets, Algorithm 3 maintains only two EVMDDs. As a result, the cost of reaching a goal state is directly contained in the EVMDD $\mathcal{E} \overset{\max}{\wedge} \mathcal{E}_{\mathcal{G}}$. Plan

reconstruction works similarly to the version for BDDs, with the difference that one checks if a predecessor is in the monolithic EVMDD closed list with corresponding costs. If no goal state was found, the states represented in $\mathcal{E}$ are expanded and the successor states are added to the open list (line 9). Finally, in line 10, we remove all closed states from the open list by computing the max of the current open list and the complement of the closed list $\neg closed$.

For a detailed description of other symbolic search variants, we refer the interested reader to Torralba (2015) and Speck et al. (2018a).

3. Experimental Setup

In this article, we extend the basic SAS$^+$ formalism with three expressive extensions—conditional effects, axioms, and state-dependent action costs—first in isolation and then in combination, resulting in four main sections. Each of these sections contains a theoretical analysis of symbolic search with the extensions and an empirical comparison to other state-of-the-art techniques.

For the empirical evaluations, we collected an extensive set of benchmark tasks from the literature. To determine the required model extensions for each task, we used the Fast Downward translator component to translate the first-order PDDL tasks into a grounded representation (Helmert, 2009), using up to 30 minutes and 8 GiB. Based on the grounded representation, we collected the PDDL planning tasks that have conditional effects, axioms, and/or state-dependent action costs *after* a successful translation phase. We then divided them into benchmark sets for the different model extensions and their combination. We provide detailed information about the resulting benchmark sets and the planners we compare against in the dedicated sections.

For our symbolic search algorithms we use six variants, using forward, backward and bidirectional search with BDDs and/or EVMDDs. The algorithms introduced in Section 2 work for all extensions, provided that the planning tasks can be represented using BDDs and EVMDDs. Thus, to show that all the algorithms guarantee to return an optimal plan, it suffices to show that the transition relations and goal correctly encode the semantics of the planning task with each of the extensions.

Our BDD-based and EVMDD-based symbolic searches are performed using extended versions of SymK (Speck, Mattmüller, & Nebel, 2020) and Symple (Speck et al., 2018b), and support finite-domain variables (Edelkamp & Helmert, 1999; Helmert, 2009) and state-dependent action costs in $\mathbb{N}_0$. Both planners are based on Fast Downward (Helmert, 2006) and have their origin in SymBA* (Torralba et al., 2014). For the BDD representation we use the CUDD library (Somenzi, 2015) and for the EVMDD representation we use MEDDLY (Babar & Miner, 2010). To empirically compare state-of-the-art planning systems, we apply the following optimizations to BDD-based symbolic search with CUDD (not available in MEDDLY): the relational product for image and preimage computation (Burch et al., 1994), and in bidirectional search, we impose initial time and BDD size limits of 1 minute and 10 million nodes per search step in both directions, doubling these limits if both directions exceed the limits in the current step. In each of the experiments presented in the different sections, we impose the same resource limits and use the same hardware. For each planner run, we allocate 30 minutes and a memory limit of 8 GiB and use Downward Lab (Seipp,

Pommerening, Sievers, & Helmert, 2017) to run our experiments on Intel Xeon Gold 6130 CPUs. All our code, the code for other planners, the benchmarks, and the experiment data are available online (Speck et al., 2024).

4. Conditional Effects

In classical planning, actions are typically assumed to have fixed, context-independent effects. However, it is often desirable to specify effects that are context-dependent, i.e., based on the state in which the action is applied. *Conditional effects* are an extension to planning formalisms like SAS^+, and they provide a natural and compact way to model such contextual effects (Pednault, 1989; Nebel, 2000; Helmert, 2009). It is well-established that conditional effects contribute to the expressive power of the planning formalism. Nebel (2000) demonstrated this by showing that conditional effects cannot be compiled away if the length of plans is limited to grow only linearly (and the size of the model is not allowed to grow exponentially). Conditional effects can be compiled away if one allows the resulting plans to grow polynomially. Sometimes, this compilation can even be done efficiently, by exploiting specific structures of conditional effects (Gerevini, Percassi, & Scala, 2024; Percassi, Scala, & Gerevini, 2024). However, in general the resulting tasks often pose a significant challenge to planning systems, since the search space usually grows exponentially with the search depth.

Recognizing the importance of conditional effects, many modern planners provide support for this model extension, including cost-optimal planning systems. This unusually high support for a model extension beyond SAS^+ can be attributed to the fact that this feature has become a mandatory requirement for the most recent installments of the Classical Track at the International Planning Competition (i.e., in 2014, 2018 and 2023). In these competitions, a subset of the benchmark domains include conditional effects, and alternative formulations without the feature are not provided, in contrast to previous IPCs (e.g., IPC 2004; Hoffmann & Edelkamp, 2005).

The history of symbolic planners supporting conditional effects natively already started with the first symbolic search planner, the Model Checking Integrated Planning System (MIPS; Edelkamp & Helmert, 2001). MIPS was designed to support the full action description language (ADL; Pednault, 1989), which includes conditional effects. While there have been some symbolic planners without support for conditional effects, the feature is almost universally supported in modern symbolic planning systems. However, to the best of our knowledge, there is no comprehensive description and analysis of how conditional effects can be implemented efficiently and compactly in symbolic search planners. A notable exception is the planner abstract by Kissmann et al. (2014), which describes the encoding of conditional effects in the Gamer planner.

In this section, we fill this literature gap and describe in detail how modern symbolic search planners support conditional effects by efficiently encoding them in transition relations. At the end of the section, we present an empirical study that compares symbolic and explicit search planners on domains that feature conditional effects. Our results show that overall symbolic search achieves comparable performance while often complementing other state-of-the-art techniques.

4.1 Formalism

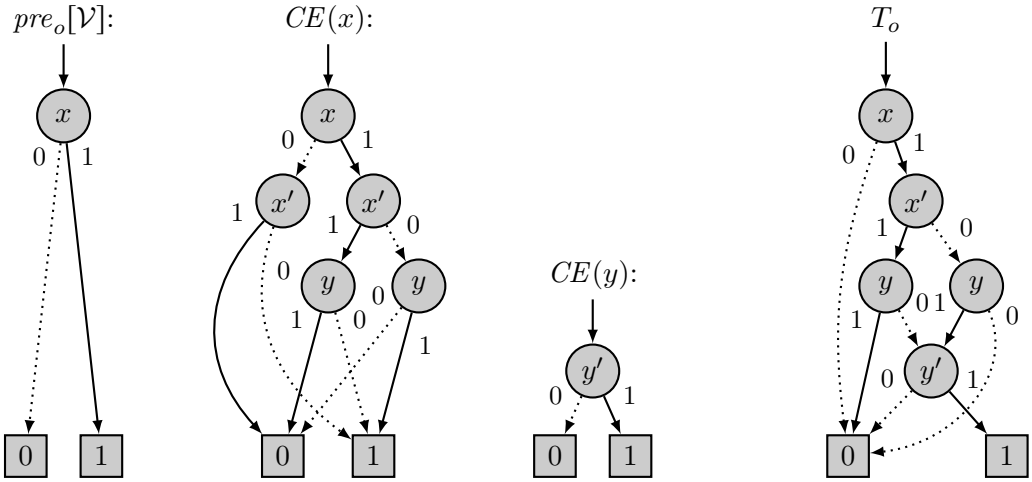
A planning task with conditional effects is defined as follows (Pednault, 1989; Helmert, 2009).

Definition 8 (Planning Task with Conditional Effects). A *planning task with conditional effects* $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$ is identical to a SAS$^+$ planning task (Definition 1), except that the effects eff_o of an operator $o = \langle \mathit{pre}_o, \mathit{eff}_o \rangle \in \mathcal{O}$ are not partial variable assignments, but a set of *conditional effects* $(\mathit{cond} \rhd v = d)$, where $v \in \mathcal{V}$, $d \in D_v$, and the *effect condition* cond is a partial state. Effects must be well-formed, i.e., the conditions of multiple conditional effects that assign different values to the same variable can never hold in the same state.[4] As for SAS$^+$ tasks, an operator $o \in \mathcal{O}$ is applicable in a state s iff pre_o is satisfied in s, i.e., $\mathit{pre}_o \subseteq s$. The result of applying operator o to a state s is the state $s' = s[\![o]\!]$, where for all $v \in \mathcal{V}$ we have $s'(v) = d$ if there exists $(\mathit{cond} \rhd v = d) \in \mathit{eff}_o$ and $\mathit{cond} \subseteq s$, and $s'(v) = s(v)$ otherwise. Slightly abusing notation, for an operator o, $\mathcal{V}(\mathit{eff}_o)$ is the set of all variables affected by the effects of o.

Example 4. Consider the Mars rover planning problem described in Example 1. Consider an extended goal description that includes downlinking the scientific data collected from the rock sample. However, the rover can only establish a connection to downlink the data at certain locations, and such an operation is costly. Therefore, the objective is to send as much data as possible in a single downlink. Further, assume that the rover can send all collected data at location (0,7). Such an operator *downlink*-0-7 can be easily modeled with conditional effects. It has a precondition that the rover is at location (0,7) and conditional effects that specify the corresponding data upload when the rover has collected a rock sample at a certain location. So we have two conditional effects: if the rock sample at (5,1) has been collected, the corresponding data will be downlinked, and the same is true for the rock sample at (7,1). Thus, if the rover has not yet collected any rock samples, the operator has no effect (apart from inducing costs). If it has collected the rock sample at (5,1), this data will be communicated, and the same is true for (7,1). If both are collected, both sets of data are downlinked. Assigning a cost of 1 to the *downlink*-0-7 operator, the optimal plan for this extended example is similar to the one described in Example 1. However, we now have the additional *downlink*-0-7 operator, which is not executed until both rock samples have been collected on the way to the final location.

In Example 4, one could introduce operators for each possible subset of the collected data, resulting in four operators with corresponding preconditions, splitting the context in which they can be applied. This corresponds to a combinatorial compilation of introducing new operators for each context. However, this approach can incur an exponential number of operators, which usually makes it infeasible to solve the task in practice. This issue becomes apparent for Example 4, when there are not just two, but n rock samples to collect and downlink. In this case, the SAS$^+$ task without conditional effects needs $O(2^n)$ operators.

4. If an operator has conflicting effects, an "equivalent" conflict-free version can be generated, as is done in the Fast Downward translator (Helmert, 2009), which we use in the presented symbolic search planners.



(a) BDDs representing the relevant components of the transition relation T_o, where $CE(v) = CE^+(v) \vee CE^+(v)$.

(b) BDD representing the transition relation T_o.

Figure 4: Visualization of BDDs representing the components and the final symbolic representation T_o of an operator $o = \langle \{x\}, \mathit{eff}_o \rangle$ with one conditional and one unconditional effect $\mathit{eff}_o = \{(y \rhd \neg x), (\varnothing \rhd y)\}$.

4.2 Symbolic Search

The underlying idea for supporting conditional effects in symbolic search is to encode them directly in the transition relation. As for SAS$^+$ tasks, this implies creating a transition relation for each operator $o \in O$ that assigns the value 1 to each predecessor-successor state pair $\langle s, s' \rangle$ induced by o if and only if o is applicable in state s, and $s' = s[\![o]\!]$. All other state pairs are assigned the value 0.

The main idea for encoding an operator $o \in \mathcal{O}$ with conditional effects, is to collect the conditions *cond* under which a certain effect d occurs for variable v. The induced conditional effect can be encoded as $CE^+(v) = \bigvee_{(cond \rhd v=d) \in \mathit{eff}_o} (cond[\mathcal{V}] \wedge (v', d))$. This formula reflects that if any condition with v as the effect variable is true in the resulting state, then the effect is encoded using the primed variable v'. In addition, we need to encode the frame axiom if none of the conditions are met to set a value for the variable v. This can be represented as $CE^-(v) = (\bigwedge_{(cond \rhd v=d) \in \mathit{eff}_o} \neg cond[\mathcal{V}]) \wedge (v \iff v')$.

Using the functions CE^+ and CE^-, we can now define a transition relation for an operator $o \in \mathcal{O}$ with conditional effects as follows.

$$T_o = pre_o[\mathcal{V}] \wedge \bigwedge_{v \in \mathcal{V}(\mathit{eff}_o)} (CE^+(v) \vee CE^-(v)) \wedge \bigwedge_{v \in \mathcal{V} \setminus \mathcal{V}(\mathit{eff}_o)} (v \iff v') \tag{1}$$

For operators without conditional effects, i.e., those where all effect conditions are empty, Equation (1) reduces to the one in Definition 5 since CE^+ is $\mathit{eff}_o[\mathcal{V}']$, and CE^- is always false.

Example 5. Consider a set of two propositional variables $\mathcal{V} = \{x, y\}$ and an operator $o = \langle pre_o, \mathit{eff}_o \rangle$ with $pre_o = \{x\}$ and $\mathit{eff}_o = \{(y \rhd \neg x), (\varnothing \rhd y)\}$. The transition relation

representing o consists of the following components. The precondition $pre_o[\mathcal{V}] = x$ specifies that x must be true for applying o. For variable x, we derive $CE^+(x) = y \wedge \neg x'$ and $CE^-(x) = \neg y \wedge (x \iff x')$, specifying that x becomes false if y is true, and retains its value if $\neg y$. For variable y, we get $CE^+(y) = \top \wedge y' = y'$ and $CE^-(y) = \bot \wedge (y \iff y') = \bot$, encoding the unconditional effect y' (making y true). In this example, all variables are part of an effect, so the last part of Equation (1) is an $\bigwedge$ over the empty set and evaluates to true. Combining these components we get $T_o = x \wedge ((y \wedge \neg x') \vee (\neg y \wedge (x \iff x'))) \wedge y'$. Formula T_o precisely describes operator o, considering that 1) the precondition x must be satisfied, 2) if y is true, then $\neg x'$ holds, and if $\neg y$, then x retains its value, and 3) y becomes true in the subsequent state. Figure 4 illustrates T_o and its components as BDDs.

It is important to note that, unlike in the case of SAS$^+$ operators without conditional effects, the BDD/EVMDD representation of the transition relation of a single operator T_o is not guaranteed to have a polynomial size relative to the size of the operator o. Specifically, $CE^+(v)$ encodes an arbitrary DNF formula, whose representation as one of these decision diagrams may be exponential. An example is an operator that makes a variable true if two variables representing adjacent cells of a grid are true, which is known to result in exponentially large decision diagrams regardless of the order of the variables (Edelkamp & Kissmann, 2011). However, in many cases the BDD representation is concise in practice. In Example 4, where the compilation to the SAS$^+$ formalism would involve an exponential number of operators, the BDD representation remains polynomial as it exploits the independence between conditional effects where conditions and effects impact only a single variable. Thus, by directly constructing the transition relation using Equation (1), we can completely avoid this exponential blow-up.

4.3 Empirical Evaluation

We compare different planning algorithms on 11 domains with conditional effects obtained from the literature (McDermott et al., 1998; Koehler & Schuster, 2000; Hoffmann & Edelkamp, 2005; Palacios & Geffner, 2009; Haslum, 2011, 2013; Vallati, Chrpa, Grześ, McCluskey, Roberts, & Sanner, 2015; Segovia-Aguas, Jiménez, & Jonsson, 2018). The benchmark set consists mainly of domains (primary and alternative versions) from the 2004, 2014, and 2018 International Planning Competitions. These domains range from developing strategies in popular board games to planning outdoor activities. In addition, "non-IPC domains" are included, ranging from a compilation of conformant to classical planning (Palacios & Geffner, 2009) to finite-state controller synthesis (Segovia-Aguas et al., 2018).

To compare our implementations of BDD-based and EVMDD-based symbolic search, which embed conditional effects directly in the transition relation, we ran a non-symbolic (blind) search in the form of A* with the blind heuristic h^0 and the $h^{\max}$ heuristic (Bonet & Geffner, 1999) as a baseline. In addition, we selected state-of-the-art cost-optimal planning systems from the International Planning Competition 2018. Specifically, we chose the top three non-portfolio planners from the optimal track of the 2018 International Planning Competition on domains with conditional effects. These planners are Scorpion, Complementary-2 and Metis-1.

Scorpion (Seipp, 2018) performs A* (Hart, Nilsson, & Raphael, 1968) with an admissible heuristic (Pearl, 1984) based on component abstraction heuristics combined with saturated

	A*		IPC 2018			EVMDD			BDD		
Domain	h^0	$h^{\max}$	Compl-2	Metis-1	Scorpion	fw	bw	bd	fw	bw	bd
Briefcaseworld (50)	7	8	10	9	**16**	7	5	7	8	6	9
Caldera (78)	24	20	26	**33**	26	20	8	20	29	22	28
Cavediving (17)	**4**	**4**	**4**	**4**	**4**	**4**	0	**4**	**4**	**4**	**4**
Citycar (40)	10	19	13	**21**	15	12	0	10	19	7	19
Flashfill (15)	0	**2**	0	1	0	0	0	0	**2**	0	1
FSC (57)	**19**	**19**	4	**19**	**19**	5	0	5	7	0	7
GED (26)	20	20	20	20	**24**	14	8	13	20	8	20
Miconic-Simple (150)	81	78	138	144	147	123	103	123	**150**	149	**150**
Nurikabe (38)	16	16	16	16	**19**	12	4	10	17	8	17
Settlers (40)	8	9	9	9	**10**	2	0	1	9	0	9
T0 (120)	28	30	33	**40**	37	24	17	24	**40**	23	39
Total Coverage (631)	217	225	273	316	**317**	223	145	217	305	227	303
Norm. Coverage (11)	3.43	3.78	3.79	4.48	**4.56**	2.96	1.45	2.79	4.27	2.51	4.22

Table 1: Number of problems solved within each of the 11 domains with **conditional effects** and neither axioms nor state-dependent action costs. The total coverage is the sum of all solved instances, while the normalized coverage is the aggregated percentage of solved instances per domain. The number of problems grounded within resource limits per domain is indicated in brackets.

cost partitioning (Seipp & Helmert, 2018) to find optimal plans. For tasks with conditional effects, the abstraction heuristics are pattern database heuristics for systematically and explicitly generated patterns consisting of 1, 2, and 3 variables (Pommerening, Röger, & Helmert, 2013).

Complementary-2 (Franco et al., 2018) implements an explicit A* search with pattern database (PDB) heuristics. This planner sequentially generates new PDBs that complement the previous ones (Franco, Torralba, Lelis, & Barley, 2017). The heuristic values for these PDBs are computed and stored using symbolic data structures, specifically using BDDs and symbolic search. The support for conditional effects is based on the approach we have outlined in this article, which involves encoding them directly in the transition relation. However, the main difference between standard symbolic search and Complementary-2 is the use of symbolic search: In Complementary-2, it is primarily used to compute a heuristic value. The heuristic value is derived by multiple symbolic backward searches on simplified versions of the task described by the considered patterns and then combined using 0-1 cost partitioning.

Metis-1 (Sievers & Katz, 2018) is an explicit heuristic search planner that implements orbit space search (Domshlak, Katz, & Shleyfman, 2015) for symmetry-based pruning and a

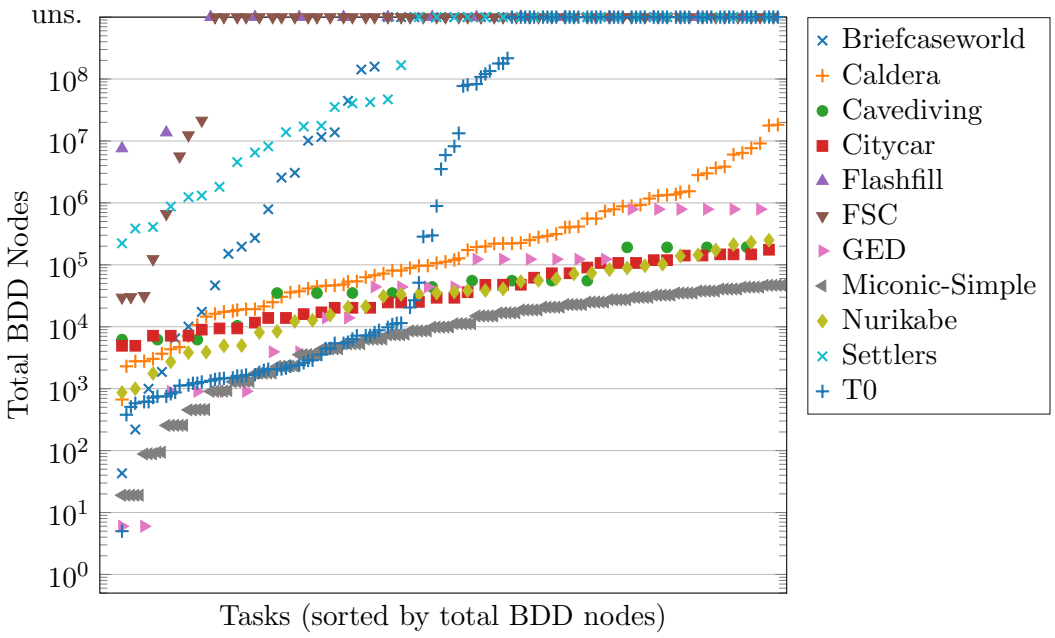


Figure 5: Visualization showing the cumulative size of **transition relations** represented by **BDDs** for planning tasks with **conditional effects**. The y-axis shows the summed nodes of the transition relations T_o, which represent the individual operators $o \in \mathcal{O}$ within the planning tasks. Tasks within each domain are arranged in ascending order based on the total number of BDD nodes and are distributed along the x-axis.

variant of the LM-cut heuristic (Helmert & Domshlak, 2009) with context splitting (Röger, Pommerening, & Helmert, 2014) to support conditional effects.

Table 1 shows the number of solved instances per domain (coverage) for different approaches. When considering symbolic search, we observe that BDD-based symbolic search generally outperforms EVMDD-based symbolic search. This superiority can be attributed to the more efficient operations and libraries for BDD manipulation, coupled with the fact that the structural advantage of compactly representing cost functions does not amortize well when costs lack sufficient diversity. By lack of diversity in the cost function, we mean that the cardinality of the range of the cost function $\mathcal{C}$ is small, i.e., $|range(\mathcal{C})|$ is small. This can result in only a modest structural advantage of EVMDDs over BDDs, given the limited additive separability of these functions. Extreme cases are domains with unit operator costs where $|range(\mathcal{C})| = 1$, such as Briefcase, FSC, Miconic-Simple, Nurikabe, and T0.

Notably, on this benchmark set, forward search proves to be the overall best performing search direction for symbolic search. While it is commonly reported in practice that bidirectional search performs best (Torralba et al., 2017; Speck et al., 2020), it is also known that its effectiveness is domain dependent, as observed here for both BDD- and EVMDD-based search. A reason for this is that, in domains where actions have many conditional effects, the transition relations may be complex to represent (see Figure 5). While this affects the search in both directions, regression is more affected, possibly because the conditions of conditional effects are more complex than the effects themselves. In fact, in the three

domains with larger transition relations (Flashfill, FSC and Settlers), backward search is unable to solve any instance.

Comparing symbolic search to explicit A* with the blind and hmax heuristics, we observe that both forward and bidirectional BDD-based searches perform favorably overall, while EVMDD-based searches perform less favorably. Figure 5 shows the cumulative sizes of the transition relations representing individual operators in the form of BDDs for the eleven domains considered. The main observation is that symbolic search is often empirically better than or equal to explicit search when it is possible to create the transition relations. This is typically the case for domains with a moderate number of operators and conditional effects, such as Miconic-Simple. However, in domains such as FSC (finite-state controller), where the number of conditional effects is vast (thousands per operator), symbolic search approaches tend to run out of memory when creating the necessary data structures. This trend continues and is especially true for the less memory-optimized symbolic search approach based on EVMDDs.

Looking at the top three planners on domains with conditional effects from IPC 2018, we find that the explicit heuristic search approaches Metis-1 and Scorpion perform the best overall. A key reason for their performance is their ability to handle domains with numerous conditional effects more effectively. However, we can also see that symbolic search has complementary strengths compared to these explicit heuristic search approaches. For example, symbolic forward BDD search outperforms Scorpion in five domains, and vice versa. Interestingly, while Metis-1 performs worse than Scorpion overall, we can observe that it is more similar to symbolic search. It performs better than symbolic forward BDD search in four domains, while it performs worse in two domains.

When comparing symbolic search with Complementary-2, we observe that pure forward and bidirectional symbolic search with BDDs outperform Complementary-2's symbolic pattern database approach overall. The same pattern holds when examining the number of domains where forward symbolic and bidirectional symbolic search with BDDs outperform Complementary-2: 7 versus 1. The main reason for this is that Complementary-2 faces similar challenges when dealing with planning problems containing numerous conditional effects. The approach first generates all transition relations, as in pure symbolic search, and then abstracts them to create simpler tasks.

Overall, in this empirical evaluation, we find that symbolic search performs overall comparably, albeit weaker, than other modern approaches based on explicit heuristic search. It excels in domains involving moderately many conditional effects, but struggles to represent and deal with problems with very complex actions where the transition relation is hard to represent, especially when the actual search is not challenging (e.g., due to plans being relatively short). In summary, symbolic search is complementary to other search strategies when it comes to planning tasks involving conditional effects.

5. Complex Action and Goal Conditions via Axioms

In classical planning, Boolean or finite domain variables are commonly used to describe the states of the world, the preconditions and the effects of actions, and the goal criteria (Fikes & Nilsson, 1971; Bäckström & Nebel, 1995). For many planning problems, however,

complex action preconditions or goals are desirable or even necessary for a compact task description (Hoffmann & Edelkamp, 2005; Thiébaux et al., 2005).

Axioms allow to model such complex preconditions and goals compactly by introducing a set of derived variables whose values are not directly influenced by the actions but are derived from the values of other variables using a set of logical rules. Thiébaux et al. (2005) showed that axioms are an essential feature, i.e., it is not possible to compile them away in a compact and efficient way in general. In particular, such compilations can lead to a super-polynomial increase in either plan length or description size.[5]

Although axioms are an essential modeling feature, modern planning techniques rarely support them, especially not in cost-optimal planning. Most admissible heuristics commonly used within A* search, one of the most prominent approaches to cost-optimal planning, are not defined for tasks with axioms. The few heuristics that do support axioms are based on naïve relaxations that treat axioms as zero-cost actions, which may greatly reduce the accuracy of the heuristics. One exception are axiom-aware delete relaxation heuristics, obtained by applying a model for state constraints to planning with axioms (Ivankovic & Haslum, 2015; Haslum, Ivankovic, Ramírez, Gordon, Thiébaux, Shivashankar, & Nau, 2018). While these heuristics are often informative, they are also time-consuming to compute and therefore often do not pay off in terms of coverage or runtime.

In this section, we define planning with axioms and describe how to extend symbolic search algorithms to support axioms natively, based on the work by Speck et al. (2019). Our empirical study on different planning domains shows that the symbolic axiom encodings yield an optimal planner that compares favorably with other state-of-the-art methods that support axioms.

5.1 Formalism

A planning tasks with axioms extends a SAS$^+$ planning task (Definition 1) with a set of derived variables and a set of axiom rules (Thiébaux et al., 2005; Helmert, 2008).

Definition 9 (Planning Task with Axioms). A *planning task with axioms* is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{D}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, extending a SAS$^+$ planning task with a set of *axioms* $\mathcal{A}$ and *derived/secondary* variables $\mathcal{D}$. Each derived variable d is a binary variable with domain $D_d = \{0, 1\}$ and a default value of 0. In addition to the set $\mathcal{S}$, which contains all states defined over the primary variables $\mathcal{V}$, we let $\mathcal{S}_E$ refer to the set of all *extended* states defined over $\mathcal{V} \cup \mathcal{D}$. For a state $s \in \mathcal{S}$, we let $\mathcal{A}(s) \in \mathcal{S}_E$ denote the corresponding extended state. Preconditions pre_o of operators $o \in \mathcal{O}$ and the *goal condition* $\mathcal{G}$ are defined over primary and secondary variables $\mathcal{V} \cup \mathcal{D}$. In contrast, effects eff_o of operators $o \in \mathcal{O}$ and the *initial state* $\mathcal{I}$ are defined only over the primary variables $\mathcal{V}$. An operator is applicable in a state $s \in \mathcal{S}$ if $pre_o \subseteq \mathcal{A}(s)$ and a state $s \in \mathcal{S}$ is a goal state if $\mathcal{G} \subseteq \mathcal{A}(s)$. We discuss the semantics of operator application below.

An *axiom (rule)* $r \in \mathcal{A}$ has the form $head \leftarrow body$ where the head $head$ is a value assignment of 1 to a derived variable $d \in \mathcal{D}$, i.e., $head = \langle d, 1 \rangle$ (or just $head = d$), and the body $body$ is a partial state over primary and secondary variables $\mathcal{V} \cup \mathcal{D}$. Given an axiom

5. Thiébaux et al. (2005) consider planning tasks described in PDDL (McDermott et al., 1998; Hoffmann & Edelkamp, 2005), while we consider planning tasks, and in particular axioms, in a grounded and normalized form.

rule r, we denote with $body(r)$ and $head(r)$ the body or head of r, respectively. The set of axioms is partitioned into a totally ordered set of axioms layers $\mathcal{A}_1 \prec \cdots \prec \mathcal{A}_k$. The layer of an axiom is defined by the layer of its head, which is determined by a partition of the set of derived variables into subsets $\mathcal{D}_1 \prec \cdots \prec \mathcal{D}_k$. We assume that this partition forms a *stratification*, i.e., that for all $i = 1, \ldots, k$, and for each $d_i \in \mathcal{D}_i$, it holds that (1) if $d_j \in \mathcal{D}_j$ appears in the body of an axiom with head d_i, then $j \leqslant i$, and (2) if $d_j \in \mathcal{D}_j$ appears as $\neg d$ (with its default value) in the body of an axiom with head d_i, then $j < i$.

The semantics of axioms is as follows: (1) to evaluate a derived variable, only axioms in the current or previous layers have to be considered, and (2) axioms have negation-as-fault semantics, i.e., if a fact cannot be derived as true, it is assumed to be false in subsequent layers. Given a state $s \in \mathcal{S}$ (over $\mathcal{V}$), the extended state $\mathcal{A}(s) \in \mathcal{S}_E$ (over $\mathcal{V} \cup \mathcal{D}$) is uniquely defined by the standard stratified semantics (Apt, Blair, & Walker, 1988; Thiébaux et al., 2005). In other words, axioms are evaluated in a layer-by-layer fashion using fixed-point computations. More precisely, given a state $s \in \mathcal{S}$, first all derived variables $d \in \mathcal{D}$ are set to their default value 0. Second, a fixed-point computation is performed for each axiom layer in sequence to determine the final values of the derived variables (Helmert, 2008). Algorithm 4 describes the axiom evaluation algorithm for explicit states.

Algorithm 4: Axiom evaluation for explicit states (Helmert, 2008).

```
Data: Axiom layers A_1 ≺ ··· ≺ A_k, state s ∈ S
Result: Extended state s' = A(s)
foreach primary variable v ∈ V do
    s'(v) := s(v)
foreach derived variable d ∈ D do
    s'(d) := 0
foreach axiom layer i = 1, ..., k do
    while there is an axiom head ← body ∈ A_i such that s' ⊨ body and s' ⊭ head do
        s'(head) := 1
return s'
```

Intuitively, the axioms form a background theory that makes it possible to capture/derive some properties of states, i.e., secondary variables, from the primary state variables. While an operator can only change the values of the primary variables directly (eff_o is a partial variable assignment over $\mathcal{V}$), the values of the secondary variables are derived from the axioms and the primary variables in the successor state.

Example 6. Consider the *navigate* operators of Example 1, which navigate the rover between cells with a cost of 0. This verbose modeling of the rover navigation leads to a larger state space and longer plans than necessary. Instead, reachability can be expressed as a recursive property with axioms and derived properties. To model a *navigate* operator with axioms that moves the rover to a reachable location *loc*, we introduce a derived variable *reachable-loc* as a precondition for the *navigate* operator. The values for the derived variables *reachable-loc* are determined by a single layer of axioms $\mathcal{A}_1$, containing the following axioms for each location *loc* and each adjacent location *from*:

- *reachable-loc* ← *rover-at* = *loc*
- *reachable-loc* ← *free-loc* ∧ *reachable-from* ∧ *adjacent-from-loc*

Intuitively, the first axiom encodes that the current location of the rover is reachable. The second axiom means that a free location adjacent to a reachable location is also reachable. By applying the axioms until a fixed point is reached, the current state s is expanded to $\mathcal{A}(s)$, which then contains the information about the actual reachable locations (*reachable-loc*) based on the current state s, which contains the rover location and all relevant information about the map and what locations are currently free. This encoding provides a natural and concise modeling of the transitive closure property of reachability. Moreover, it provides not only a smaller state space (as actions moving the robot can consider only the relevant locations where rock samples can be taken ignoring all intermediate locations that are traversed during the navigation), but also a shorter plan $\pi = \langle$*navigate*-7-2, *sample-rock*-7-1, *navigate*-5-1, *sample-rock*-5-1, *navigate*-0-5$\rangle$, which avoids the irrelevant choice of the exact path the rover must take.[6]

In Example 6, the reachable cells for the rover are always the same (as no action has an effect on *free-loc*), but this is not necessarily true in general. For example, it could be the case that actions by the rover make some cells impassable. A similar scenario where reachability can change from state to state is Sokoban, where the locations of the boxes affect the reachability of the cells (Ivankovic & Haslum, 2015; Miura & Fukunaga, 2017).

5.2 Symbolic Search

Speck et al. (2019) introduced three sound and complete variants to support axioms in symbolic search. While none of the three encodings dominates the others in theory, their *symbolic translation* approach has several advantages over the other two methods. It readily allows forward, backward, and bidirectional search and, as we will show below, can be straightforwardly combined with other encodings of expressive planning features. Furthermore, it has been shown to be the empirically dominant strategy (Speck et al., 2019). For these reasons, in this article we will consider only the symbolic translation approach to supporting axioms with derived variables and compare it to other state-of-the-art approaches to cost-optimal planning with axioms. The symbolic translation encoding is based on precomputing a symbolic representation over the primary variables $\mathcal{V}$ for each derived variable $d \in \mathcal{D}$.

Definition 10 (Primary Representation). Let $d \in \mathcal{V} \cup \mathcal{D}$ be a (primary or derived) variable and $\mathcal{A}$ a set of axioms. The *primary representation* of d is the set of states S_d, which contains all states over $\mathcal{V}$ where d is evaluated to true, i.e., $S_d = \{s \in \mathcal{S} \mid \mathcal{A}(s) \models d\}$.

Clearly, it is infeasible to construct the primary representation of a derived variable by enumerating all the states in which it is true. Therefore, we create the primary representation in the form of its characteristic function using decision diagrams as the underlying data structure. Algorithm 5 describes a fixed-point algorithm that builds the primary representations of all variables $d \in \mathcal{D}$. We compute the primary representations of derived

6. If the navigation operators of the rover would have non-zero costs, such a modeling would require state-dependent action costs, which we discuss in Section 6.

Algorithm 5: Construction of primary representations (Speck et al., 2019).

Data: Axiom layers $\mathcal{A}_1 \prec \cdots \prec \mathcal{A}_k$
Data: Derived variables $\mathcal{D}_1 \prec \cdots \prec \mathcal{D}_k$ partitioned into layers
Result: Symbolic primary representations χ_{S_d}, $d \in \mathcal{D}$

```
1  foreach D_1 ≺ ··· ≺ D_k do
2      foreach d ∈ D_i do
3          χ_{S_d} ← ⋁_{r ∈ A_d^{<i}} body(r)[χ_{S_D}/D]
4      queue ← D_i
5      while queue is not empty do
6          d ← queue.pop()
7          foreach head ← body ∈ A_i with d ∈ body do
8              χ_{S_head} ← χ_{S_head} ∨ body[χ_{S_D}/D]
9              if χ_{S_head} has changed then
10                 queue.insert(head)
11 return {χ_{S_d} | d ∈ D}
```

variables layer by layer, which is possible because the primary representation χ_{S_d} of each derived variable $d \in \mathcal{D}$ depends only on previous layers.

In each iteration, Algorithm 5 begins by gathering information for derived variables that relies solely on variables from lower levels. More precisely, we initialize the set of states in which $d \in \mathcal{D}$ is true by considering all axioms in $\mathcal{A}_d^{<i}$, where $\mathcal{A}_d^{<i}$ stands for all axiom rules that have d in their heads and whose bodies contain only variables that are either primary variables or derived variables of lower levels, i.e., $\mathcal{A}_d^{<i} = \{head \leftarrow body \in \mathcal{A} \mid head = d \text{ and } \forall d' \in \mathcal{V}(body) \cap \mathcal{D} : d' \in \mathcal{D}_j \text{ for some } j < i\}$. For an axiom r, we compute $body(r)[\chi_{S_\mathcal{D}}/\mathcal{D}]$, where the notation $\varphi[\chi_{S_\mathcal{D}}/\mathcal{D}]$ stands for the result of simultaneously replacing each derived variable $d \in \mathcal{D}$ with χ_{S_d} within the formula φ. The algorithm then proceeds to apply axioms based on variables in the same layer until a fixed point is reached. It is important to note that at no point a derived variable appears within χ_{S_d}. Algorithm 5 is sound and complete, i.e., it returns a characteristic function χ_{S_d} (in form of a DD) which precisely characterizes the primary representation S_d for each $d \in \mathcal{D}$ (Speck et al., 2019).

Example 7 (Speck et al., 2019). Consider a planning task with primary variables x, y and derived variables a, b, c, together with the following axioms:

$$r_{11} : a \leftarrow b$$
$$r_{12} : b \leftarrow \neg x$$
$$r_{13} : b \leftarrow y$$
$$r_{21} : c \leftarrow (\neg a \wedge \neg b),$$

where $\mathcal{A}_1 = \{r_{11}, r_{12}, r_{13}\}$ and $\mathcal{A}_2 = \{r_{21}\}$, i.e., $\mathcal{D}_1 = \{a, b\}$ and $\mathcal{D}_2 = \{c\}$. We derive the primary representations χ_{S_a}, χ_{S_b}, and χ_{S_c} using Algorithm 5. The first step is to consider $\mathcal{D}_1$ and derived variable a (line 3). We observe that $\mathcal{A}_a^{<1} = \varnothing$, since the body of r_{11} contains b, which is in the same layer. Consequently, the value of χ_{S_a} is $\bot$. We then continue with b

and find that $\mathcal{A}_b^{<1}$ consists of r_{12}, r_{13}, so $\chi_{S_b} = \neg x \vee y$. We then continue by initializing the queue (line 4) with both a and b. After popping a from the queue, we encounter a situation where there is no rule in the current layer that contains a in the body (line 7). As a result, we move on to pop b. In this case, we have a single rule in the current layer with b in the body, specifically the rule $r_{11} = a \leftarrow b$. We set $\chi_{S_a} = \bot \vee \chi_{S_b}$, which is equivalent to $\neg x \vee y$ (line 8). We need to re-insert a into the queue. However, as with the previous iteration, we can immediately remove it from the queue and exit the loop. This marks the end of the iteration for $\mathcal{D}_1$, resulting in $\chi_{S_a} = \chi_{S_b} = \neg x \vee y$. In the subsequent iteration, we consider c, and the set $\mathcal{A}_c^{<2}$ contains only one rule, namely $r_{21} = c \leftarrow (\neg a \wedge \neg b)$. Consequently, we derive

$$\begin{aligned}\chi_{S_c} &= (\neg\chi_{S_a} \wedge \neg\chi_{S_b}) \\ &= (\neg(\neg x \vee y) \wedge \neg(\neg x \vee y)) \\ &\equiv x \wedge \neg y.\end{aligned}$$

Note that due to the equivalence of χ_{S_a} and χ_{S_b}, a DD library can potentially bypass intermediate transformations, since both DDs represent the same object. The algorithm stops now because there are no rules with c in the body. As a result, we arrive at the primary representations: $\chi_{S_a} = \chi_{S_b} = \neg x \vee y$ and $\chi_{S_c} = x \wedge \neg y$.

The key idea of the symbolic translation encoding is to replace all occurrences of derived variables in the planning task with their corresponding primary representation. In particular, derived variables in operator preconditions and the goal formula are replaced by their corresponding primary representation, resulting in a new set of operators $\mathcal{O}'$ and a new goal formula $\mathcal{G}'$, i.e., $\mathcal{O}' = \{\langle pre_o[\chi_{S_\mathcal{D}}/\mathcal{D}], \mathit{eff}_o\rangle \mid o \in \mathcal{O}\}$ and $\mathcal{G}' = \mathcal{G}[\chi_{S_\mathcal{D}}/\mathcal{D}]$. More precisely, if a DD for φ is constructed and a derived variable $d \in \mathcal{D}$ occurs in φ, we replace d with its primary representation in the form of another DD χ_{S_d} during the construction. This uses the compact and efficient nature of DDs and results in a symbolic representation of the transition relations and the goal formula based on the primary representations of derived variables. With the symbolic compilation we obtain a planning task without axioms and derived variables such that no reasoning about derived variables during the actual search is required. Therefore, the symbolic compilation offers the possibility of forward, backward and bidirectional search. Since $\mathcal{A}(s) \models d$ iff $s \models S_d$, it follows that this encoding is sound and complete, i.e., a sequence of actions π of a planning task Π found using the symbolic translation encoding is a plan iff π is a plan for Π (Speck et al., 2019).

Note that this symbolic translation encoding is different from compilations that convert SAS$^+$/PDDL with axioms to SAS$^+$/PDDL without axioms (Gazen & Knoblock, 1997; Thiébaux et al., 2005), since at no point it creates an explicit version of the compiled task (a new SAS$^+$/PDDL representation). While the primary representation, and thus the symbolic translation, can lead to exponential size growth in the worst case, in practice the concise representation of decision diagrams alleviates this problem.

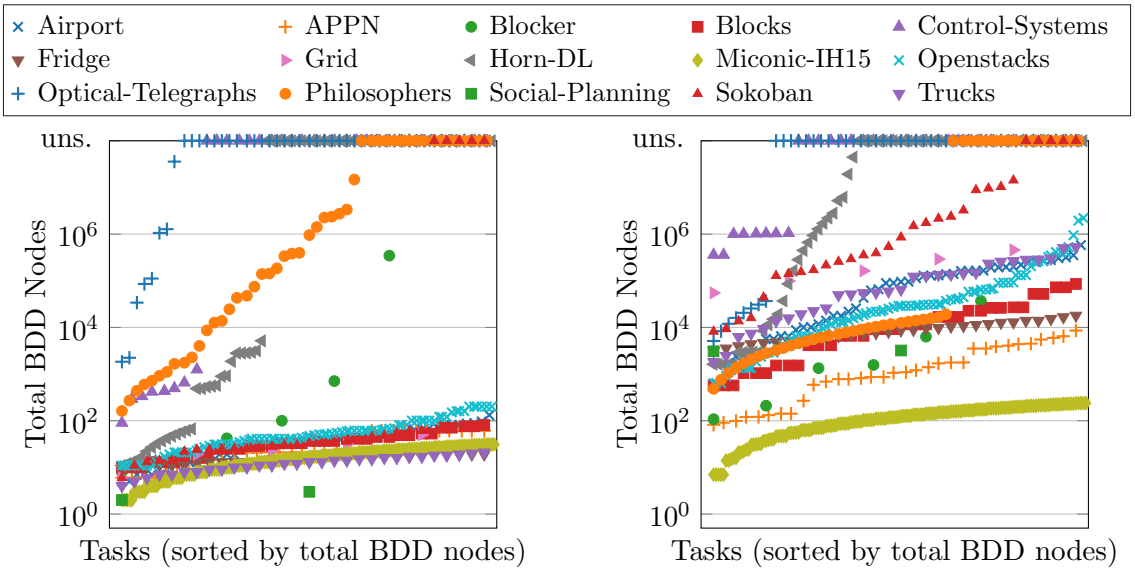
5.3 Empirical Evaluation

We compare different planning algorithms, including symbolic search, on 15 different domains with axioms collected from the literature (Koehler & Schuster, 2000; Hoffmann &

	A*					EVMDD			BDD		
Domain	h^0	$h^{\max}$	$h^{\max}_{\text{full}}$	$h^{\max}_{\text{3val}}$	h^{PDB}	fw	bw	bd	fw	bw	bd
Airport (50)	19	**22**	11	18	14	14	11	14	20	11	20
APPN (33)	11	11	9	10	8	11	10	11	**22**	18	**22**
Blocker (7)	**7**	**7**	5	6	**7**	6	6	6	6	6	6
Blocks (35)	18	18	11	17	28	20	18	29	21	21	**30**
Control-Systems (35)	**16**	14	3	7	15	7	2	7	7	7	7
Fridge (30)	0	0	0	0	0	0	0	0	**1**	0	**1**
Grid (5)	1	**3**	1	2	2	1	1	1	1	1	**3**
Horn-DL (75)	**57**	53	36	51	15	22	19	23	21	17	20
Miconic-IH15 (150)	60	60	45	55	52	125	145	145	**150**	**150**	**150**
Openstacks (85)	38	36	15	32	38	49	29	48	**64**	45	**64**
Optical-Telegraphs (48)	2	2	1	2	1	2	0	2	**4**	0	**4**
Philosophers (48)	5	5	3	5	4	5	3	6	**12**	4	**12**
Social-Planning (2)	**2**	**2**	1	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
Sokoban (30)	24	27	5	21	**28**	21	21	23	24	25	25
Trucks (30)	8	**10**	3	6	6	6	4	6	9	5	9
Total Coverage (663)	268	270	149	234	220	291	271	323	364	312	**375**
Norm. Coverage (15)	6.70	7.18	3.61	6.08	6.38	6.18	5.60	6.67	7.42	6.47	**8.10**

Table 2: Number of problems solved for each of the 15 domains with **axioms** and neither conditional effects nor state-dependent action costs. The total coverage is the sum of all solved instances, while the normalized coverage is the sum over the percentages of solved instances per domain. The number of problems grounded within resource limits per domain is shown in brackets.

Nebel, 2001; Edelkamp, 2003a; Hoffmann & Edelkamp, 2005; Ghosh, Dasgupta, & Ramesh, 2015; Ivankovic & Haslum, 2015; Borgwardt, Hoffmann, Kovtunova, Krötzsch, Nebel, & Steinmetz, 2022). The benchmark set contains several (alternative) formulations of domains with axioms from previous International Planning Competitions, ranging from verification problems such as Philosophers or Optical Telegraphs, via puzzle games such as Sokoban, to practical applications such as Airport ground control. The benchmark set also includes "non-IPC domains", such as repairing a Fridge or planning with state constraints formalized in Horn Description Logics (Horn-DL). Note that some domains do not contain inherently defined axioms (e.g., Trucks), but complex preconditions with quantifiers that are translated into axioms in a way that the considered planning systems can handle (Helmert, 2009). Some domains in our benchmark set have a single layer with rather simple axioms, while others, such as the Blocker domain by Ivankovic and Haslum (2015), have complex recursive axiom structures with up to four layers. The explicit A* search (Hart et al., 1968) is evaluated with the blind heuristic h^0, the (naïve) max heuristic $h^{\max}$, the full ASP consistency checking max heuristic $h^{\max}_{\text{full}}$, the 3-valued relaxed max heuristic $h^{\max}_{\text{3val}}$, and the canonical PDB axiom-aware heuristic h^{PDB} (default and recommended pat-



(a) Number of **BDD** nodes representing the **goal description**.

(b) Summed number of **BDD** nodes for the **transition relations** T_o, representing the individual operators $o \in \mathcal{O}$ of the planning tasks.

Figure 6: Sizes of the critical BDDs required to represent planning tasks with **axioms** along the y-axis. Tasks within each domain are sorted in ascending order based on the total number of BDD nodes and distributed along the x-axis.

tern selection: one PDB per primary variable), which are all admissible for planning with axioms (Ivankovic & Haslum, 2015).

For symbolic search, the computation of the primary representation for each derived variable is generally not a limiting factor. In most instances, the precomputation, as described in Algorithm 5, only takes a few seconds. However, there are situations where the computation of the underlying BDD is infeasible due to the large number of derived variables and axioms. For example in the Control-Systems domain, where instances have up to 602 757 derived variables and 2 411 000 axioms, this results in a comparatively low coverage for symbolic search (Table 2).

After computing the primary representation of each derived variable, the next step is to compute the BDDs representing the goal and/or transition relations. This requires combining the representation of all the derived variables mentioned in the goal and/or operator's preconditions. In the worst-case, the size of the resulting BDD grows exponentially in the number of derived variables that are combined. So, even in cases where the representation of each derived variable is not large, this may become a bottleneck. Figure 6 shows the size of the BDDs representing the goal and operators of the task. Considering Figure 6a, it becomes apparent that in some domains, such as Philosophers or Optical Telegraphs, the substitution of derived variables in the goal formula becomes computationally challenging or even infeasible for larger instances. This is related to some cases where it is known that the BDDs representing the set of goal states are exponential in the size of the task regardless of the variable ordering (Edelkamp & Kissmann, 2008). For other domains, such as

Horn-DL or Sokoban, incorporating the primary representations of the derived predicates in the preconditions into the transition relations becomes infeasible as the task size increases (Figure 6b).

Considering the coverage results shown in Table 2, BDD-based symbolic search comes out on top in terms of overall coverage. However, we observe that explicit heuristic search and symbolic search have complementary strengths and perform better in different domains. In particular, the (naïve) $h^{\max}$ heuristic outperforms symbolic bidirectional search with BDDs in six domains, while symbolic bidirectional search outperforms the (naive) $h^{\max}$ heuristic in seven domains. Explicit blind search can sometimes perform better than symbolic search, including BDD-based search, in domains with many derived variables and axioms, such as Control-Systems, Blocker, and Horn-DL, where there are tasks with more than 10 000 derived variables and computing the primary representations of the transition relations is impractical. Then, there are some domains (Airport, Sokoban, and Trucks) where despite symbolic search being superior to explicit search without heuristics, some heuristics can get better performance. Interestingly, in these domains the transition relations are also large, so this is again an important factor to decide whether symbolic search is to be preferred to A*.

The empirical performance of symbolic search is particularly strong in domains where the representation of the transition relations is small and does not scale exponentially with increasing task size (Figure 6b). In such domains, symbolic search typically outperforms all explicit-search variants. Interestingly, this also includes the two domains where the goal representation is impractical (Optical-Telegraphs and Philosophers). Even though the size of the goal representation negatively affects backward search configurations (as they start their computation from the goal), forward search is less affected by the size of the goal representation.

In the comparison of EVMDD and BDD based representations, we see that EVMDD-based search is inferior to BDD-based search in the domains at hand, primarily because its structural advantages over BDDs do not provide significant benefits when representing predominantly propositional formulas with non-diverse action costs. In this benchmark set domains have mostly unit costs, except Openstacks and Sokoban where costs are either 0 or 1. In Section 7 we also consider domains with axioms and richer state-dependent action costs.

6. State-Dependent Action Costs

In classical planning, it is common to assume constant, context-independent action costs (Fikes & Nilsson, 1971; Bäckström & Nebel, 1995). Similar to not having context-dependent effects (cf. Section 4), this often results in increased effort for the modeler. When a planning problem inherently involves context-dependent action costs, or more precisely, state-dependent action costs (sometimes called *conditional costs*), the modeler must distribute these costs over multiple copies of the original action. In addition, the structure of the original cost function is hidden, which, however, could provide useful information and more compact representation possibilities for planning algorithms (Geißer, 2018). If we consider, e.g., probabilistic planning in the form of factorized Markov decision processes (Puterman, 1994), state-dependent action costs/rewards have been the standard for a long time (Sanner,

2010; Geißer, 2018; Geißer, Speck, & Keller, 2020) and are supported by many different approaches (Keller & Eyerich, 2012; Geißer & Speck, 2018; Cui & Khardon, 2018). Therefore, recently there has been increased interest in classical planning with state-dependent action costs (SDACs) (Keller & Geißer, 2015; Keller, Pommerening, Seipp, Geißer, & Mattmüller, 2016; Geißer, 2018; Corraya, Geißer, Speck, & Mattmüller, 2019; Mattmüller et al., 2018; Ivankovic, Gordon, & Haslum, 2019; Haslum et al., 2018; Drexler, Seipp, & Speck, 2021).

Speck et al. (2021) showed that SDACs increase expressiveness of classical planning by proving that they cannot be compiled away. Specifically, there is no compilation scheme that preserves plan length linearly, when the cost function is in FP. Moreover, it is impossible to preserve the plan length polynomially when the cost function is in FPSPACE, unless the polynomial hierarchy collapses to the third level. Since many practically relevant cost functions are in FP (all cost functions in this article fall in this class), it is often necessary to support SDACs natively in algorithms. This avoids the polynomial blowup in terms of domain description size and/or original plan length caused by compilation schemes.

In the following, we formally introduce planning with SDACs and show how it is possible to natively represent SDACs with decision diagrams to perform symbolic search (Speck et al., 2018b, 2018a; Speck, 2022). Our empirical evaluation shows that the native support of SDACs within symbolic search can be beneficial compared to other explicit search approaches that are partially based on compilations.

6.1 Formalism

A planning task with state-dependent action costs (SDACs) is defined as follows (Geißer et al., 2015; Geißer, 2018; Speck, 2022).

Definition 11 (Planning Task with SDACs). A *planning task with state-dependent action costs* $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{I}, \mathcal{G} \rangle$ is identical to a SAS$^+$ planning task (Definition 1), except that it has a *state-dependent action cost function* $\mathcal{C} : \mathcal{O} \times \mathcal{S} \to \mathbb{N}_0$, where the cost of applying an operator $o \in \mathcal{O}$ depends on the state $s \in \mathcal{S}$ in which it is applied. A plan $\pi = \langle o_0, \ldots, o_{n-1} \rangle$ for an SDACs planning task that generates a sequence of states $s_0, \ldots, s_n$ generalizes a constant-cost plan (Definition 2) by considering for the cost computation the state in which each operator is applied, i.e., $\mathcal{C}(\pi) = \sum_{i=0}^{n-1} \mathcal{C}(o_i, s_i)$. With $\mathcal{C}_o : \mathcal{S} \to \mathbb{N}_0$ we refer to the local SDAC function for an operator $o \in \mathcal{O}$ induced by $\mathcal{C}$.

In general, a state-dependent cost function can have an arbitrary form and even be uncomputable (Geißer, 2018; Speck, 2022). In practice, however, it is useful to restrict the form and expressiveness of the cost function. Similar to previous works, we mainly consider cost functions that can be evaluated in polynomial time and have a concise form.

Definition 12 (Operator Cost Function). We define a language $\mathcal{L}$ by the following Backus normal form:

$$t ::= c \mid v \mid t + t \mid t - t \mid t \cdot t \mid |t|,$$

where $c \in \mathbb{Z}$, $v \in \mathcal{V}$. The semantics of $\mathcal{L}$ is defined for operators $o \in \mathcal{O}$ and states $s \in \mathcal{S}$ as follows:

$$\begin{aligned}
&\mathcal{C}_o^c(s) = c \\
&\mathcal{C}_o^v(s) = s(v) \\
&\mathcal{C}_o^{t \circ t'}(s) = \mathcal{C}_o^t(s) \circ \mathcal{C}_o^{t'}(s) \qquad \text{for } \circ \in \{+, -, \cdot\} \\
&\mathcal{C}_o^{|t|}(s) = |\mathcal{C}_o^t(s)| = \begin{cases} \mathcal{C}_o^t(s) & \text{if } \mathcal{C}_o^t(s) \geqslant 0 \\ -\mathcal{C}_o^t(s) & \text{otherwise} \end{cases}
\end{aligned}$$

For a given term $t \in \mathcal{L}$, the interpretation $\mathcal{C}_o^t$ specifies the operator cost function of operator $o \in \mathcal{O}$, where we restrict the allowed operator cost function to a positive range, i.e., $\mathcal{C}_o^t : \mathcal{S} \to \mathbb{N}_0$. In the following, we often identify a cost function $\mathcal{C}_o^t$ with the term t that defines it.

We emphasize that a SAS$^+$ planning task (Definition 1) is an important special case, in which operators have constant costs, i.e., the costs are independent of the state in which the operator is applied. Modeling operator costs as state-dependent allows for a more natural and concise representation of planning problems, as the following example illustrates.

Example 8. Consider the *navigate* actions of the rover in Example 1. Now assume that we want the *navigate* action to have a cost of 1, plus additional costs that depend on the weight of the carried rock samples. Without SDACs (Definition 1), we need a distinct *navigate* action connecting two cells for each possible carrying scenario: when the rover carries no rock sample, one sample, the other sample, or both samples simultaneously. With SDACs, we can model such actions in a natural and concise way by specifying the cost function of the *navigate* action. Suppose we have two binary variables, *carries-sample*-5-1 and *carries-sample*-7-1 that have value 1 iff the rover carries the rock sample from location $(5, 1)$ or $(7, 1)$, respectively. Then the SDACs of the *navigate* action is $1 + [\![\textit{carry-sample}\text{-}5\text{-}1]\!] \cdot 2 + [\![\textit{carry-sample}\text{-}7\text{-}1]\!] \cdot 5$,[7] assuming that carrying the rock sample from $(5, 1)$ costs 2, while carrying the sample from $(7, 1)$ costs 5 due to its heavier weight.

Unfortunately, SDACs is not an official PDDL feature. Previous research has traditionally used the grounded input format of the Fast Downward planner (Helmert, 2009) with finite-domain variables to describe planning tasks with SDACs. We have extended our planners to directly support SDACs specified as per Definition 12, but modeled directly in PDDL to improve practical usability. The following example demonstrates how to model SDACs in PDDL by considering the navigation costs of the rover in our running example.

Example 9. Figure 7a shows an excerpt of a PDDL domain describing Example 8. The *:cost* section in the *navigate* action describes the state-dependent action costs. The + is an n-ary operator that describes the addition of n elements (similarly, there are $-$ and $*$ for subtraction and multiplication). In this example, we have two elements connected by the + operator. The first, *navigate-cost*, describes the navigation cost of the rover in question. The second element uses the *Summation (sum-over)* operation, which sums over all *rock_samples*

7. For readability, we sometimes use Iverson brackets $[\![x]\!]$, which are defined to take the value 1 if x is true, and 0 otherwise.

```
(:functions
    (total-cost) - number
    (weight ?s - rock-sample) - number
    (navigate-cost ?r - rover) - number
)

(:action navigate
    :parameters (?r - rover ?from ?to - location)
    :precondition (and
        (at ?r ?from)
        (connected ?from ?to)
    )
    :effect (and
        (not (at ?r ?from))
        (at ?r ?to)
    )
    :cost
    (+
        (navigate-cost ?r)
        (sum-over
            (?s - rock-sample)
            (* (carries ?r ?s) (weight ?s))
        )

    )
)
```

(a) Functions and *navigate* action with state-dependent action cost modeled in a PDDL domain.

```
(:objects
    r1 - rover
    s1 s2 - rock-sample

    ... ; omitted parts

)

(:init
    (= (total-cost) 0)
    (= (navigate-cost r1) 1)
    (= (weight s1) 2)
    (= (weight s2) 5)

    (at r1 loc-07-03)
    (at s1 loc-05-01)
    (at s2 loc-07-01)

    ... ; omitted parts

)
```

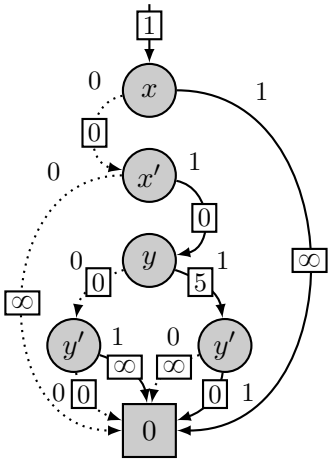
(b) Objects and initial state with numeric constants in a PDDL problem.

Figure 7: Excerpts of a PDDL model describing the *navigate* action illustrated in Examples 8 and 9 with state-dependent action costs depending on whether a rover is carrying rock samples and their weight.

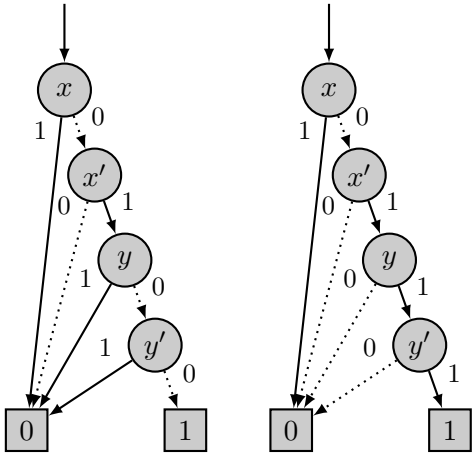
?s and adds the cost of the different rock samples ?s if they are currently carried by the rover ?r. Note that the syntax of the *Sum (sum-over)* and *Product (product-over)* operators is similar to the standard PDDL quantifiers *exists* and *forall*. Finally, the specific values of the navigation costs of the rovers and the specific weights of the rock samples are defined as functions in the problem description (Figure 7b). Of course, it is also possible to use a number directly in the domain description, e.g., if all rovers have the same navigation cost.

6.2 Symbolic Search

While we can represent reachability costs in tasks with SDACs using both BDDs and EVMDDs, the latter representation is more natural, so we discuss it first. In contrast to symbolic search with BDDs, where costs are represented by partitioning sets of states into subsets with identical cost values (see Example 2), with EVMDDs we can represent a set of reachable states together with reachability costs simultaneously. More precisely, we assign sets of reachable states $S \subseteq \mathcal{S}$ the corresponding cost $g \in \mathbb{N}_0$ and unreachable states the cost ∞. The same idea can be applied to transition relations T_o, which represent an operator $o \in \mathcal{O}$. Here, the function $\chi_{T_o} : \mathcal{S} \times \mathcal{S} \to \mathbb{N}_0 \cup \{\infty\}$ maps all pairs of states $\langle s, s' \rangle$ to $\mathcal{C}_o(s)$ if o is applicable in s and $s' = s[\![o]\!]$, and all other state pairs are mapped to cost ∞. The construction of the transition relation is analogous to that of a state-independent cost operator (Definition 5). Example 10 illustrates the idea of encoding an operator with SDACs as an EVMDD (and as BDDs). By encoding SDACs directly into the transition



(a) An EVMDD representing the transition relation of operator o and the cost $\mathcal{C}_o(s)$.



(b) Two BDDs representing the transition relation of operator o, where the left BDD encodes the cost of 1 and the right BDD encodes the cost of 6.

Figure 8: Visualization of different variants for a symbolic representation of an operator $o = \langle \neg x, x \rangle$ with state-dependent action costs $\mathcal{C}_o(s) = 5y + 1$.

relation this way, it is possible to readily perform a complete and optimal forward, backward and bidirectional symbolic search with EVMDDs (Speck et al., 2018a).

In addition to the EVMDD-based symbolic search for SDACs planning tasks, multiple BDDs can be used to support SDACs (Speck, 2022). The underlying idea is to first create an ADD (instead of an EVMDD) that represents each operator as a transition relation that includes costs, before partitioning the ADD into multiple BDDs. In other words, we create multiple transition relations for each operator with SDACs, one for each possible cost value, encoding as a precondition the corresponding condition for the cost. This approach may result in multiple transition relations with different costs, but allows us to perform the standard symbolic search with cost bucketing, using the full power of sophisticated BDD operations and libraries. Disassembling the ADD into multiple BDDs is feasible in polynomial time (Torralba, 2015). Thus, when the transition relation, along with the cost function, can be compactly represented as an ADD, it can also be represented compactly with multiple BDDs. The union of these BDDs accurately represents the transition relation of an operator o, excluding the SDACs of the operator $\mathcal{C}_o$, and distinct cost values are associated with the disassembled BDDs. Therefore, using these transition relations in standard forward, backward, or bidirectional symbolic search ensures the optimality.

Example 10. Consider a set of two propositional variables $\mathcal{V} = \{x, y\}$ and an operator $o = \langle \neg x, x \rangle$ with SDACs $\mathcal{C}_o(s) = 5y+1$ for $s \in \mathcal{S}$. Figure 8 visualizes the transition relation of o represented with different decision diagrams, where the precondition (predecessor states) are encoded with unprimed variables and the effects (successor states) with primed variables. Figure 8a represents the EVMDD representing the transition relation of o, while encoding the cost $\mathcal{C}_o(s)$ for each state $s \in \mathcal{S}$ at the same time. To represent operator o with BDDs, multiple BDDs are necessary to decompose the cost function. Figure 8b illustrates the two

BDDs representing the two possible cost values of $\mathcal{C}_o(s)$, specifically 1 and 6. The left BDD encodes the cost of 1 by adding $\neg y$ to the precondition. Note that this change incurs that $\neg y$ also holds in all successor states, i.e., $\neg y'$, since variable y does not occur in the effect (closed world assumption). Similarly, the right BDD encodes y as a precondition to obtain a cost of 6.

The performance of symbolic search algorithms is greatly influenced by the choice of decision diagram and their ability to represent the relevant functions (goal, transition relations, and sets of states reached during the search) compactly.

One of the main advantages of using EVBDDs/EVMDDs over ADDs and BDDs is that they can represent certain additive functions exponentially more compactly. To illustrate this, consider Example 8 as one increases the rock samples the rover can carry. With one rock sample (of weight 5), this is exactly the same as in Figure 8. And increasing the number of rock samples increases the size of the EVMDD linearly, adding three nodes per rock sample and encoding the weight of the rock sample locally (in the same way as Figure 8a between nodes y and y'). Thus, for cost functions that are simply linear combinations, EVMDDs can always represent them efficiently. However, the ADD/BDD representation may require exponential size in the number of rock samples to represent such a function, depending on the weights of the rock samples. The reason is that we need to construct a different BDD (or a different terminal node in the ADD) for each possible summed weights. And, in the worst-case, there are 2^n possible sums for n rock samples, e.g., if the weight of sample i is 2^i.

Using BDDs instead of the structurally more expressive decision diagrams has the advantage that BDDs offer more advanced and sophisticated optimizations. For example, complement edges can be used to store BDDs more compactly (Brace et al., 1990), and there are more efficient operations available for manipulating BDDs (Burch et al., 1994).

In general, however, representing the cost function as a decision diagram can lead to exponential size in the worst case, regardless of the type of decision diagram used. An example is a cost function that depends on whether vertically or horizontally adjacent cells of a grid have been visited, which imposes an exponentially large decision diagram in the size of the grid, regardless of the variable order (Edelkamp & Kissmann, 2011).

6.3 Empirical Evaluation

We compare different planning algorithms (Table 3) on 7 different domains with SDACs collected from the literature (Geißer, 2018; Corraya et al., 2019; Speck et al., 2021; Speck, 2022). The domains are mostly modified versions of well-known domains from previous International Planning Competitions, extended with state-dependent action costs. We compare explicit A* search (Hart et al., 1968) with various translation-based heuristics (Geißer, 2018) and forward, backward and bidirectional symbolic search using EVMDDs (Speck et al., 2018a) or BDDs (Speck, 2022). In explicit A* search, the cost function is represented as an EVMDD, which is used to evaluate the actual costs (g-values) (Geißer, 2018). We consider the blind heuristic (h^0) and two versions of the $h^{\max}$ heuristic (Bonet & Geffner, 1999): $h^{\max}_{\text{c}}$ uses the combinatorial translation (Geißer, 2018) and $h^{\max}_{\text{dd}}$ uses the EVMDD translation (Geißer et al., 2015) to compute the heuristic values. The $h^{\max}_{\text{dd}}$ heuristic uses EVMDDs to represent the cost function, which is then compiled into the (relaxed) plan-

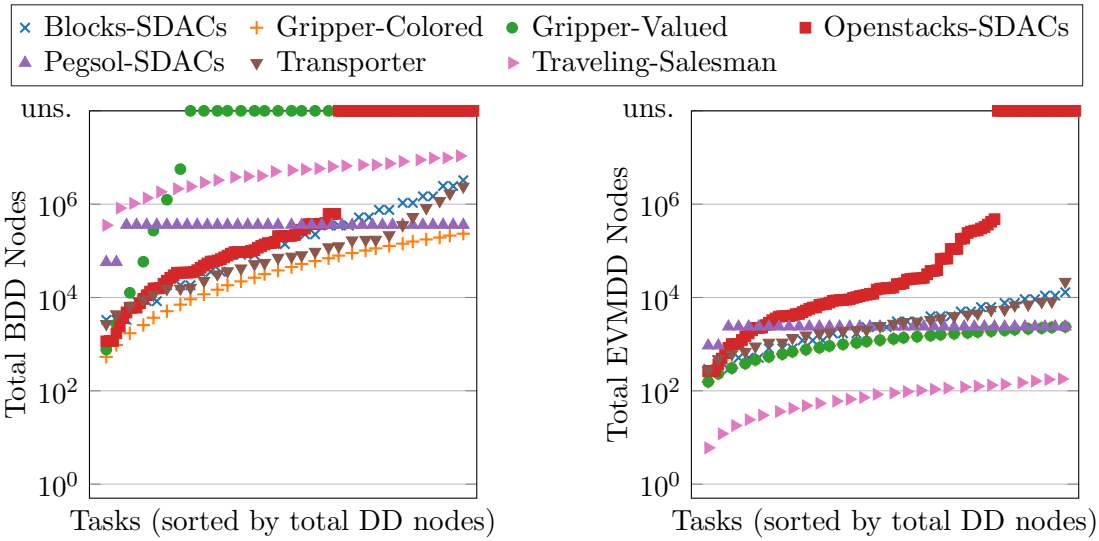
	A*			EVMDD			BDD		
Domain	h^0	$h^{\max}_{\mathrm{c}}$	$h^{\max}_{\mathrm{dd}}$	fw	bw	bd	fw	bw	bd
Blocks-SDACs (35)	21	9	18	15	9	15	18	18	**27**
Gripper-Colored (30)	8	4	8	12	8	12	**20**	13	**20**
Gripper-Valued (30)	9	4	10	8	5	**19**	7	5	7
Openstacks-SDACs (110)	13	13	13	43	32	41	**59**	51	57
Pegsol-SDACs (36)	**35**	0	**35**	33	8	33	**35**	13	**35**
Transporter (30)	24	19	24	24	23	**29**	24	23	**29**
Traveling-Salesman (26)	**22**	12	20	15	10	15	18	10	18
Total Coverage (297)	132	61	128	150	95	164	181	133	**193**
Norm. Coverage (7)	3.91	1.73	3.77	3.79	2.36	4.30	4.41	3.08	**4.82**

Table 3: Number of problems solved within each of the 7 domains with **state-dependent action costs** and neither conditional effects nor axioms. The total coverage is the sum of all solved instances, while the normalized coverage is the aggregated percentage of solved instances per domain. The number of problems grounded within resource limits per domain is indicated in brackets.

ning graph for heuristic computation. Thus, $h^{\max}_{\mathrm{dd}}$ incorporates symbolic data structures for heuristic purposes only, without any elements of symbolic search. Consequently, A* with $h^{\max}_{\mathrm{dd}}$ is a purely explicit heuristic search, using EVMDDs for heuristic computation only.

Table 3 shows the number of problems solved per domain using different search approaches. Overall, we see that native support for SDACs within symbolic search compares favorably with explicit search approaches. Comparing symbolic search with BDDs and EVMDDs, there are per-domain differences in coverage, with BDD-based symbolic search performing better in most individual domains and overall. Figure 9 shows that, in most of these domains both BDDs and EVMDDs are able to represent the cost function efficiently, the only exception being Openstacks-SDAC. It is important to note that the number of EVMDD nodes and BDD nodes are not directly comparable, as each EVMDD node requires more memory (having k instead of 2 children and information in the edges). So, in cases where both curves only differ on a constant factor (as happens in most domains here), which of the two data structures is more compact in terms of memory depends on implementation details and the exact library used.

The cost functions $\mathcal{C}$ of the domains Blocks-SDACs, Gripper-Colored, Openstacks-SDACs, and Pegsol-SDACs are simple sums over objects for which certain properties hold in a state, yielding costs bounded by the number of objects. For Transporter and Traveling-Salesman, the costs correspond to distances traveled on a map. Overall, these cost functions induce the same cost for many states. Thus, $|\mathit{range}(\mathcal{C})|$ is often small, and the domains have a low diversity of action costs. This is one explanation why EVMDDs cannot offer advantages over BDDs (Table 3).



(a) The summed **BDD** nodes of the **transition relations** T_o, representing the individual operators $o \in \mathcal{O}$ within the planning tasks.

(b) The summed **EVMDD** nodes of the **transition relations** T_o, representing the individual operators $o \in \mathcal{O}$ within the planning tasks.

Figure 9: Cumulative sizes of EVMDDs and BDDs representing the individual operators of planning tasks with **state-dependent action costs** along the y-axis. Tasks within each domain are sorted in ascending order based on the total number of BDD nodes and distributed along the x-axis.

However, the structural advantages of EVMDDs become apparent in the Gripper-Valued domain, a variant of the Gripper-Colored domain (McDermott, 2000; Geißer, 2018). This domain is similar to our running example. In Gripper-Colored, there are colored balls and rooms, and the goal is to use a gripper to move the balls from one room to another. The *pick up* and *drop ball* actions incur zero cost, whereas the *move* action induces a cost equivalent to the number of balls in a room that does not match the color of the ball. Instances scale the number of balls, but the number of rooms and colors remains constant so all instances have exactly two colors. In Gripper-Valued, each ball has an additional importance value, and the cost of the *move* action is the sum of these misplaced ball values. Specifically, each ball is assigned a cost represented by different powers of two, resulting in diverse action costs. As explained in Section 6.2, BDDs cannot represent this cost function compactly, but EVMDDs have a linear representation. In fact, Figure 9 shows how the EVMDDs representing the transition relation have exactly the same size in Gripper-Colored and Gripper-Valued (as they are independent of the "weight" assigned to each ball), but the BDD representation is heavily affected, scaling linearly for Gripper-Colored and exponentially for Gripper-Valued. Interestingly, having diverse action costs (Gripper-Colored vs. Gripper-Valued), makes the domain easier to solve for the explicit search approaches and the EVMDD-based search, as the costs give some guidance, which is a known phenomenon (Fan, Müller, & Holte, 2017).

Regarding the explicit approaches, we see that in many domains $h^{\max}_{\mathrm{c}}$ fails to perform a combinatorial translation, which often requires exponentially many operators. As a result, $h^{\max}_{\mathrm{c}}$ solves the least number of tasks. Explicit A* search with the h^0 and $h^{\max}_{\mathrm{dd}}$ heuristics

performs much better. These configurations use EVMDDs to concisely represent the cost functions and as a basis for the heuristic computation, which pays off for the tasks at hand. It turns out that the heuristic estimates of $h^{\max}_{\text{dd}}$ can sometimes help the search, but the performance of $h^{\max}_{\text{dd}}$ is still often nowhere close to symbolic blind search.

7. Classical Planning With Multiple Expressive Extensions

The previous sections demonstrated how to model planning problems more naturally and compactly by using the three model extensions under consideration: conditional effects, axioms and state-dependent action costs. In this section, we tackle the challenge of combining these extensions. While each extension captures different aspects of classical planning, there is actually a close connection between them. As a result, they often appear together in practice. However, to our knowledge this article is the first to consider and formalize all three extensions jointly, rather than just a subset of them.

Planning with conditional effects and axioms has been formalized by Helmert (2009). This includes a procedure capable of translating the full PDDL 2.2 Level 1 fragment (Fox & Long, 2003) along with all ADL features such as quantified and conditional effects, negation, disjunction, and quantification in conditions (Pednault, 1989) into a grounded finite-domain representation (FDR). This translation and grounding process is an integral part of Fast Downward (Helmert, 2006), on which our symbolic search planners are based. Later, Speck et al. (2019) introduced an approach for planning with axioms via symbolic search. They directly addressed planning tasks with conditional effects, supporting the aforementioned fragment of PDDL.

Furthermore, Mattmüller et al. (2018) demonstrated the frequent co-occurrence of conditional effects with conditional costs, namely state-dependent action costs. They conducted a mostly theoretical analysis of methods to jointly represent and handle these. In related work, Speck et al. (2018a) introduced EVMDD-based planning specifically tailored to tasks with state-dependent action costs. This approach directly accounts for conditional effects, as described in Section 4. When a planning task requires the natural and concise modeling of multiple extensions, similar considerations apply as for single extensions. Compiling them away may not be feasible without suffering a super-linear or super-polynomial increase in model size or plan length. However, whether certain extensions can be compiled away in the presence of others remains an open research question, which we will discuss in Section 8.

While previous work has shown that symbolic search can support two of the extensions simultaneously, the question is whether it can support all three model extensions at the same time. The short answer is yes, which is remarkable because support for only one of these features is quite rare in practice, and support for more than one feature is almost nonexistent in the literature on cost-optimal planning. Again, the reason why very few planners support even a single extension is that almost all optimal planners are based on heuristic search, and it is very difficult to design admissible, informative, and quick-to-evaluate heuristics that take into account any of the three model extensions.

In the following, we formally introduce planning tasks with conditional effects, axioms and state-dependent action costs, and show how to solve planning tasks described in this formalism using symbolic search. Our empirical evaluation shows that symbolic search

can be advantageous compared to the only other cost-optimal approach supporting such planning tasks, namely explicit blind search.

7.1 Formalism

By extending the SAS$^+$ formalism (Definition 1) with conditional effects and axioms, we arrive at planning tasks in finite-domain representation (FDR) (Helmert, 2009). We now extend the FDR formalism to also consider state-dependent action costs.

Definition 13 (Planning Task with Conditional Effects, Axioms, and SDACs). A *planning task with conditional effects, axioms, and state-dependent action costs* is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{C}, \mathcal{D}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. The components are identical to those defined by a SAS$^+$ planning task (Definition 1), with the three extensions discussed above (Definitions 8, 9 and 11). As for planning tasks with axioms, partial variable assignments are defined over primary and secondary variables. Consequently, *preconditions*, *effect conditions*, and *goal conditions* are defined over $\mathcal{V} \cup \mathcal{D}$ and the *state-dependent action cost function* is defined as $\mathcal{C} : \mathcal{O} \times \mathcal{S}_E \to \mathbb{N}_0$.

The combination of all three model extensions into a single formalism is a natural generalization that allows the modeling of several aspects simultaneously in a concise manner. An example of this is illustrated by the three scenarios outlined in Examples 4, 6 and 8, which one can consider all at the same time (see Example 11) using the formalism from Definition 13. Beyond that, the FDR formalism, along with our extension incorporating state-dependent action costs, also permits to include derived predicates in both the effect conditions and the operator cost functions, enabling the modeling of complex effect and cost conditions. More precisely, conditions *cond*, such as preconditions, effect conditions, or goal conditions, hold in a state s if and only if the extended state is a model for *cond*, denoted by $\mathit{cond} \subseteq \mathcal{A}(s)$. Additionally, for the state-dependent action cost function $\mathcal{C} : \mathcal{O} \times \mathcal{S}_E \to \mathbb{N}_0$, the cost of applying an operator $o \in \mathcal{O}$ depends on the extended state $\mathcal{A}(s) \in \mathcal{S}_E$, which includes the evaluated derived variables. Thus, cost functions may depend on arbitrarily complex conditions, as derived variables and axioms allow to encode any Boolean formula. Example 11 illustrates such an interaction of derived variables with conditional effects and state-dependent action costs.

Example 11. Consider the cost of the *navigate* actions in our running example (Example 1). Previously, we assumed that their cost depends on whether the rover is carrying rock samples and on their corresponding weights (Example 8). Now, let us assume that the cost depends only on whether any rock sample is carried. More precisely, if no rock sample is carried, the cost of the navigate action is 1, and otherwise it is 2 due to the need to drive more cautiously and slowly. Such a fact, whether at least one sample is carried, has a disjunctive nature and is typically represented by a derived predicate *carry-any-sample* and two axioms: *carry-any-sample* ← *carry-sample*-5-1 and *carry-any-sample* ← *carry-sample*-7-1. Now the desired cost function can be specified as $1 + [\![\mathit{carry\text{-}any\text{-}sample}]\!]$. Similarly, the derived predicate *carry-any-sample* can be part of an effect condition to indicate whether a rock sample is currently being carried by the rover.

Example 11 illustrates a relatively simple interaction between derived variables, effect conditions, and context-specific costs. However, the importance of complex effect conditions

represented by derived predicates is well-known and is captured by the FDR planning formalism by allowing derived predicates in effect conditions (Helmert, 2009).

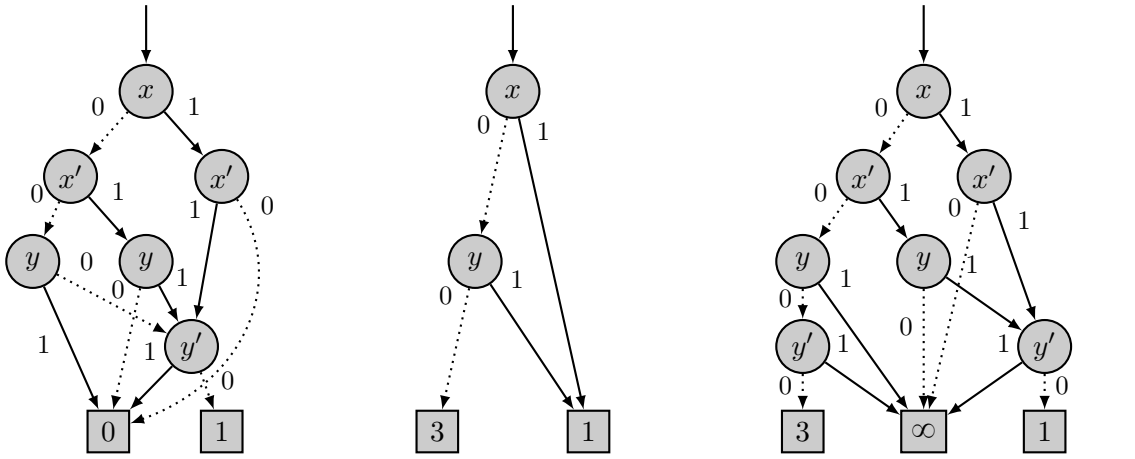
In the power supply restoration (PSR) domain (Thiébaux & Cordier, 2001), derived predicates encode whether certain devices are affected by faults, where the causes are modeled by axioms. These derived predicates are used to conditionally trigger specific effects of actions. Similarly, for state-dependent action costs, complex conditions leading to the occurrence of costs can be modeled naturally and concisely using derived predicates and axioms (Speck, 2022).

7.2 Symbolic Search

In the sections above, we showed how symbolic search can support conditional effects, derived variables with axioms, or state-dependent action costs. The underlying idea is to encode the logic of these sophisticated model extensions directly in the transition relation and/or goal condition. Interestingly, supporting all of these features simultaneously is fairly straightforward, given the ideas and concepts already introduced.

First, as outlined in Section 5, we precompute the primary representation χ_{S_d} in the form of a decision diagram for each derived variable $d \in \mathcal{D}$. Second, as described in Section 4, we create a transition relation T_o for each operator $o \in O$, possibly containing conditional effects. However, whenever a derived variable d occurs in a precondition pre_o or an effect condition $cond$ of an operator $o \in \mathcal{O}$, we replace it with the corresponding primary representation χ_{S_d}. Third, as described in Section 6, we create a decision diagram representing the cost function $\mathcal{C}_o$ of each operator $o \in O$ and add it to the operator's transition relation T_o. Again, we replace all occurrences of each derived variable d with the corresponding primary representation χ_{S_d}. Finally, we represent the initial state $\chi_{\mathcal{I}}$ and the goal states $\chi_{\mathcal{G}}$ as decision diagrams. For $\chi_{\mathcal{G}}$, we replace all derived variables with their primary representation, $\chi_{\mathcal{G}[\chi_{S_\mathcal{D}}/\mathcal{D}]}$, as described in Section 5. This gives us a symbolic representation of the planning task with conditional effects, axioms, and state-dependent action costs, and we can readily apply symbolic search.

For the individual planning extensions, we already know that the described procedure results in a symbolic representation of the planning task, which, when performing a symbolic blind search, is a sound, complete, and optimal search algorithm (Torralba et al., 2017; Speck et al., 2018a, 2019). This also applies to the combination of derived variables and conditional effects (Speck et al., 2019). Considering state-dependent action costs by adding them to the transition relation does not affect these properties. Since $\mathcal{A}(s) \models d$ if and only if $s \models S_d$ (Speck et al., 2019), it follows that for any reachable state $s \in \mathcal{S}$, the original cost of an operator $o \in \mathcal{O}$ for the extended state $\mathcal{A}(s) \in \mathcal{S}_E$ maps to the same cost value as when we replace the derived variables with their primary representation in the operator cost function $\mathcal{C}_o$ and consider s directly, i.e., $\mathcal{C}_o(\mathcal{A}(s)) = \mathcal{C}_o[\chi_{S_\mathcal{D}}/\mathcal{D}](s)$. As a result, we obtain transition relations T_o, which represent an operator $o \in \mathcal{O}$, so that the function $\chi_{T_o} : \mathcal{S} \times \mathcal{S} \to \mathbb{N}_0 \cup \{\infty\}$ maps all pairs of states $\langle s, s' \rangle$ to $\mathcal{C}_o(s)$ if o is applicable in s and $s[\![o]\!] = s'$, and all other state pairs are mapped to cost ∞. Consequently, applying symbolic (blind) search to planning tasks with conditional effects, axioms, and state-dependent action costs is a sound, complete, and optimal approach.



(a) A BDD representing the transition relation of o.

(b) An ADD representing the cost function of o.

(c) An ADD representing the combined transition relation and cost function of o.

Figure 10: Visualization of decision diagrams showing the transition relation T_o and the cost function $\mathcal{C}_o(s) = [\![\neg a]\!] \cdot 2 + 1$ for an operator $o = \langle \varnothing, \{(a \rhd x), (\varnothing \rhd \neg y)\}\rangle$ with no precondition, one unconditional and one conditional effect using the primary representation of the derived variable a, which in this example is $x \vee y$.

The following example illustrates the combination of an operator with conditional effects and a cost function defined over derived variables.

Example 12. Consider a set of two propositional primary variables $\mathcal{V} = \{x, y\}$, a set of a single propositional secondary variable $\mathcal{D} = \{a\}$, two axioms $a \leftarrow x$ and $a \leftarrow y$, and an operator $o = \langle \varnothing, \{(a \rhd x), (\varnothing \rhd \neg y)\}\rangle$ with a cost function $\mathcal{C}_o(s) = [\![\neg a]\!] \cdot 2 + 1$ for $s \in \mathcal{S}_E$. We observe that the primary representation of a is $x \vee y$, i.e., $\chi_{S_a} = x \vee y$. The transition relation with derived variables is $T_o = (a \wedge x') \vee (\neg a \wedge (x \iff x')) \wedge \neg y'$. Since we directly replace all occurrences of the secondary variable a with χ_{S_a} during construction, we get $T_o = ((x \vee y) \wedge x') \vee (\neg(x \vee y) \wedge (x \iff x')) \wedge \neg y'$. This simplifies[8] to $T_o = (x \wedge x' \wedge \neg y') \vee (\neg x \wedge ((x' \wedge y \wedge \neg y') \vee (\neg x' \wedge \neg y \wedge \neg y'))$. It implies that the operator o is always applicable, and we observe that after application y is always false ($\neg y'$) as an unconditional effect. In the case where x is true, then $\chi_{S_a} = x \vee y$, and thus the conditional effect is triggered, leaving x' true. However, if x is not true, we distinguish whether y is true. If true, then again $\chi_{S_a} = x \vee y$ is true, and x' becomes true after the application; if false, then $\neg x'$ holds. The resulting BDD representing T_o is visualized in Figure 10a. In the cost function $\mathcal{C}_o(s) = [\![\neg a]\!] \cdot 2 + 1$, we replace the occurrence of a with χ_{S_a}, yielding $[\![\neg(x \vee y)]\!] \cdot 2 + 1 = [\![\neg x \wedge \neg y]\!] \cdot 2 + 1$, which is shown as an ADD in Figure 10b. Finally, we can combine T_o and $\mathcal{C}_o$ into a single ADD representing state pairs and the costs associated with those transitions, as shown in Figure 10c. Here, transitions between two states that are not described by the operator o are mapped to a cost of infinity. As detailed in Section 6,

8. We explicitly simplify the formula for illustration purposes, but this happens automatically when encoding the formula as a BDD or EVMDD.

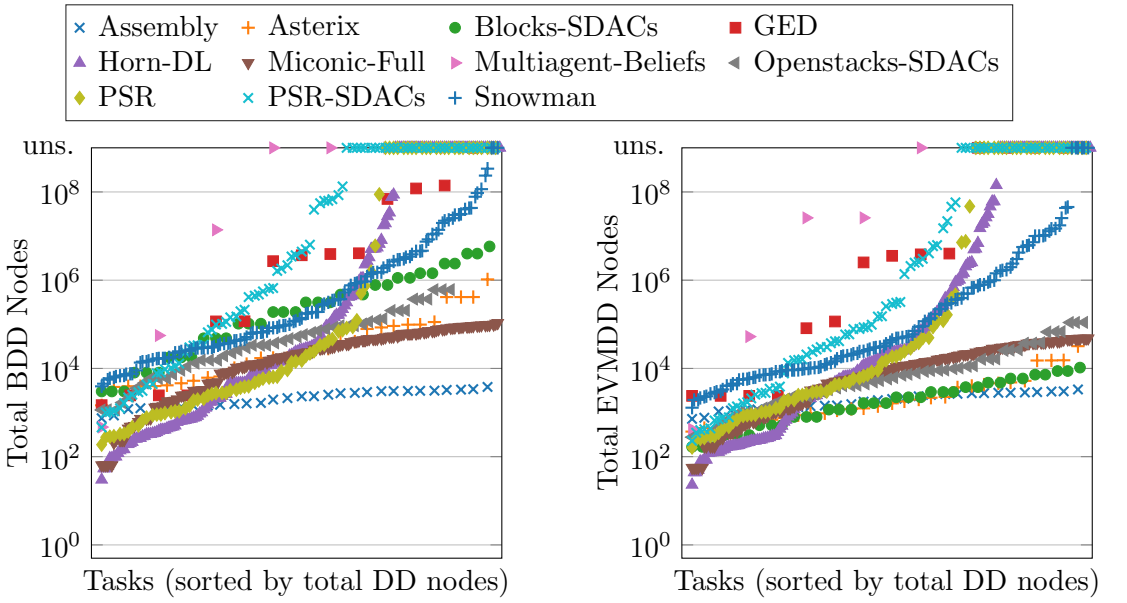
	Extensions			A*	EVMDD			BDD		
Domain	CE	Axioms	SDACs	h^0	fw	bw	bd	fw	bw	bd
Assembly (30)	✔	✔		0	4	4	5	**10**	9	**10**
Asterix (30)	✔		✔	11	**30**	29	**30**	**30**	29	**30**
Blocks-SDACs (35)		✔	✔	**19**	18	15	18	18	15	**19**
GED (14)	✔	✔		**14**	10	6	10	**14**	10	13
Horn-DL (196)	✔	✔		**175**	143	117	139	143	132	143
Miconic-Full (150)	✔	✔		78	91	68	90	**113**	88	111
Multiagent-Beliefs (7)	✔	✔		**7**	4	1	4	3	1	3
Openstacks-SDACs (55)		✔	✔	13	35	27	34	**43**	38	42
PSR (86)	✔	✔		39	59	58	60	60	60	**61**
PSR-SDACs (86)	✔	✔	✔	47	55	41	**58**	52	43	53
Snowman (123)	✔	✔		87	78	17	55	**98**	21	91
Total Coverage (812)				490	527	383	503	**584**	446	576
Norm. Coverage (11)				6.27	6.86	4.93	6.71	**7.63**	5.87	7.53

Table 4: Number of problems solved within each of the 11 domains with combinations of **conditional effects**, **axioms**, and **state-dependent action costs**, where at least two of the model extensions are present in each domain. For each domain, we indicate the extensions present: conditional effects (CE), axioms with derived predicates (Axioms), and state-dependent action costs (SDACs). The total coverage is the sum of all solved instances, while the normalized coverage is the summed percentage of solved instances per domain. The number of problems grounded within resource limits per domain is indicated in brackets.

this ADD can then be decomposed into two BDDs representing all valid (non-infinite cost) transitions between state pairs: one BDD for the transitions with cost 1 and another for those with cost 3.

7.3 Empirical Evaluation

We collected eleven planning domains from the literature, each having at least two of the three modeling features considered: conditional effects, axioms with derived predicates, and state-dependent action costs. These domains are listed with their extensions in Table 4. Most of these domains are either original or modified versions of those featured in previous International Planning Competitions. In some cases, they originate from real-world applications, such as elevator control (Koehler & Schuster, 2000), genome editing distance (GED) computation (Haslum, 2011), or power supply restoration (PSR) (Thiébaux & Cordier, 2001). In addition, certain domains stem from work on multiagent beliefs (Multiagent-Beliefs) (Kominis & Geffner, 2015), state constraints (Horn-DL) (Borgwardt et al., 2022),



(a) The summed **BDD** nodes of the **goal** and **transition relations** T_o, representing the individual operators $o \in \mathcal{O}$ within the planning tasks.

(b) The summed **EVMDD** nodes of the **goal** and **transition relations** T_o, representing the individual operators $o \in \mathcal{O}$ within the planning tasks.

Figure 11: Cumulative sizes of EVMDDs and BDDs representing the goal and the individual operators of planning tasks with combinations of **conditional effects**, **axioms**, and **state-dependent action costs** along the y-axis. Tasks within each domain are sorted in ascending order based on the total number of BDD nodes and distributed along the x-axis.

state-dependent action costs (Asterix) (Speck et al., 2018a), or modeling puzzle games (Snowman) (Bofill, Borralleras, Espasa, Martín, Patow, & Villaret, 2023).

The existing methods for finding optimal plans while supporting multiple of the considered modeling features are limited. They include explicit A* search with the blind heuristic h^0 (Geißer, 2018) and our proposed symbolic search methods using EVMDDs and BDDs (Torralba et al., 2017; Speck et al., 2018a, 2019). We compare these methods with each other.

In terms of overall coverage (Table 4), forward search with BDDs performs best. Interestingly, similar to previous empirical evaluations, bidirectional search is overall inferior to forward search. This suggests potential future work in balancing and deciding the search directions for symbolic search. Symbolic search with EVMDDs also performs well overall and even outperforms BDD approaches in the PSR-SDACs domain. A possible explanation for this is the more compact representation of complex cost functions with EVMDDs compared to BDDs, especially noticeable for larger tasks as shown in Figure 11.

A* search with the blind heuristic performs worse overall than forward and bidirectional symbolic search. However, in certain domains it is advantageous, especially when dealing with large planning problems that are not overly combinatorially challenging. This phe-

nomenon is evident in the Multiagent-Beliefs domain (Kominis & Geffner, 2015), where the grounded and simplified task involves hundreds of variables but only a handful of operators. For this kind of planning problems, it is not feasible for symbolic search approaches to create the necessary data structures to represent the planning task (Figure 11).

Despite the complementary strengths of explicit and symbolic search, we can see that on this benchmark set with multiple model features present, the symbolic search approach with the highest total coverage (forward + BDDs) solves more tasks than blind A* search in 7 out of 11 domains. In contrast, blind A* search solves more tasks than the forward+BDDs symbolic search in three domains, with a particularly large margin evident only in the Horn-DL and Multiagent-Beliefs domains. In these domains, representing goal and/or transition relations with DDs poses severe challenges as the task size increases (Figure 11).

8. Discussion and Future Work

We discussed the close connection between three extensions—conditional effects, axioms, and state-dependent action costs—for modeling planning problems concisely. Decision diagrams provide a method for representing propositional formulas and numerical functions in a compact manner, and are well suited for supporting these model extensions. The efficiency of symbolic search, however, is mainly determined by the time it takes to manipulate these decision diagrams, which in turn depends on their size. Therefore, it is important to minimize the size of these diagrams, and developing new methods for this goal is an important area of future work. Addressing this goal becomes particularly important when dealing with larger planning tasks, as highlighted in our empirical evaluations, where symbolic search faces significant challenges in certain domains.

We explored various types of decision diagrams, focusing primarily on EVMDDs and BDDs, in the context of symbolic search for classical planning. These diagrams offer a trade-off between conciseness of representation and efficiency of operation. A more detailed comparison, both in theory and in practice, could provide valuable insights into when to use which type of decision diagram. In addition, exploring other types of decision diagrams, such as (Affine) Algebraic Decision Diagrams (Bahar et al., 1997; Sanner & McAllester, 2005), Functional Decision Diagrams (Kebschull, Schubert, & Rosenstiel, 1992) or Kronecker Functional Decision Diagrams (Drechsler & Becker, 1998b), may further improve the conciseness and efficiency of the representation.

In the empirical evaluations, we saw that representing the necessary components of a planning task with decision diagrams, i.e., initial state, transition relations, and goal, can be a serious challenge in certain domains. One way to address this issue is to consider the representation of these components in a *partitioned* way. While it is common practice to have a disjunctive partitioning of the transition relations (Torralba et al., 2017), as we considered in this article and in our experiments, one could also consider a conjunctive partitioning (Burch, Clarke, & Long, 1991) for the goal representation or the transition relations. This approach could help overcome representation challenges, especially in the presence of a complex state-dependent action cost function or many derived variables and conditional effects. In particular, it will be interesting to see whether this can alleviate the representation bottlenecks of symbolic planners (Torralba, 2023; Speck, 2023; Franco, Edelkamp, & Moraru, 2023) for large planning tasks from the recent International Planning

Competition in 2023 (Taitler, Alford, Espasa, Behnke, Fišer, Gimelfarb, Pommerening, Sanner, Scala, Schreiber, Segovia-Aguas, & Seipp, 2024).

Throughout this article, we have assumed a fixed, static variable order. Since the efficiency of symbolic search is strongly influenced by the chosen variable order, and the size of decision diagrams is strongly dependent on this order, the search for good variable orders is crucial. While it has been established that computing an optimal order for decision diagrams is co-NP-complete (Bryant, 1986), the practical challenge of finding good orders remains an open question (Kissmann & Hoffmann, 2013, 2014). In addition, dynamic reordering techniques (Rudell, 1993), well studied in areas such as model checking (Yang, Bryant, O'Hallaron, Biere, Coudert, Janssen, Ranjan, & Somenzi, 1998) and logic synthesis (Scholl, Möller, Molitor, & Drechsler, 1999), have been little studied in planning (Kissmann & Hoffmann, 2014; Kissmann et al., 2014).

While the availability of accurate and efficient heuristics for the considered model extensions is limited, we have observed their potential to provide valuable guidance in certain domains. In particular, for conditional effects, these heuristics are generally competitive. Therefore, the efficient integration of heuristics into symbolic search remains an open research task. However, it is worth noting that achieving such an integration is not easy, as even a perfect heuristic can be detrimental to search performance (Speck et al., 2020). A promising family of heuristics are symbolic abstractions, which use symbolic backward search to explore an abstract state space (Edelkamp, 2002; Kissmann & Edelkamp, 2011; Torralba, Linares López, & Borrajo, 2018). The techniques presented in this paper could be the basis to obtain abstraction heuristics that can deal with all the model extensions we considered. Notably, Fišer, Torralba, and Hoffmann (2022a) introduced operator potentials based on potential heuristics (Pommerening, Helmert, Röger, & Seipp, 2015), and applied them to symbolic bidirectional heuristic search (Fišer, Torralba, & Hoffmann, 2022b). These potential functions provide a concise representation in symbolic forward search that efficiently prunes states based on their heuristic values. However, applying this idea in our setting would require computing potential heuristics for tasks with conditional effects, derived variables, and state-dependent action cost. Therefore, further research is needed to improve our understanding of search behavior in symbolic heuristic search, and whether is is possible to derive heuristics that provide size guarantees for the decision diagrams involved.

Furthermore, considering the relationship between the considered model extensions in terms of complexity and compatibility will be of interest in future investigations. It might be the case that one extension can be compiled away (with small overhead) in the presence of another extension. For example, it might be possible to simulate the evaluation of the values of the derived variables using the cost function, potentially eliminating the need to include the derived variables in the cost function. Or it may be possible to simplify a complex cost function by introducing new derived variables, thereby transferring the complexity to the evaluation of those derived variables. Similar considerations apply to conditional effects. While it is well known that conditional effects cannot be easily eliminated by concise compilations (Nebel, 2000), one can simplify the effect conditions by introducing derived predicates that encapsulate these conditions. This approach would most likely lead to more concise compiled models.

Finally, several of the considered model extensions are not yet covered by more expressive planning formalisms, such as fully observable nondeterministic planning (Cimatti, Pistore,

Roveri, & Traverso, 2003), partially observable nondeterministic planning (Bertoli, Cimatti, Roveri, & Traverso, 2006; Speck, Ortlieb, & Mattmüller, 2015), or hierarchical task network planning (Erol, Hendler, & Nau, 1996; Geier & Bercher, 2011). An analysis of how and to what extent the three extensions can enhance these planning formalisms is an important future line of research, and whether the proposed ideas can also be used for symbolic search in these settings (Kissmann & Edelkamp, 2009; Behnke & Speck, 2021; Bertoli et al., 2006).

9. Conclusions

We conducted both theoretical and empirical investigations of cost-optimal planning with three widely used model extensions: conditional effects, derived variables with axioms, and state-dependent action costs. Each of these extends different facets of classical planning while preserving the essence of the formalism. Computational complexity and compilability results from the literature underscore the need for native support for these extensions to model and solve many real-world planning problems.

This article has provided a thorough overview of symbolic search applied to classical planning with these expressive extensions. By theoretically describing and empirically analyzing the integration of symbolic search, we presented a unified framework involving Binary Decision Diagrams or Edge-Valued Multi-Valued Decision Diagrams. In addition, we introduced practical tools in the form of optimal, sound, and complete symbolic search planning algorithms and planners that provide support for planning problems involving conditional effects, derived predicates with axioms, and state-dependent action costs simultaneously. Our empirical evaluations demonstrated the complementary strengths and overall strong performance of the presented symbolic search approaches compared to other state-of-the-art methods on planning problems modeled with the considered extensions.

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) partially funded by the Swedish Research Council through grant agreement no. 2022-06725. David Speck was funded by the Swiss National Science Foundation (SNSF) as part of the project "Unifying the Theory and Algorithms of Factored State-Space Search" (UTA).

References

Apt, K. R., Blair, H. A., & Walker, A. (1988). Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pp. 89–148. Morgan Kaufmann.

Babar, J., & Miner, A. (2010). MEDDLY: Multi-terminal and Edge-Valued Decision Diagram Library. In *Proceedings of the Seventh International Conference on the Quanti-*

tative Evaluation of Systems (QEST 2010), pp. 195–196. IEEE Computer Society.

Bäckström, C., & Nebel, B. (1995). Complexity results for SAS^+ planning. *Computational Intelligence*, *11*(4), 625–655.

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., & Somenzi, F. (1997). Algebraic decision diagrams and their applications. *Formal Methods in System Design*, *10*(2–3), 171–206.

Behnke, G., & Speck, D. (2021). Symbolic search for optimal total-order HTN planning. In Leyton-Brown, K., & Mausam (Eds.), *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pp. 11744–11754. AAAI Press.

Bertoli, P., Cimatti, A., Roveri, M., & Traverso, P. (2006). Strong planning under partial observability. *Artificial Intelligence*, *170*, 337–384.

Bofill, M., Borralleras, C., Espasa, J., Martín, G., Patow, G., & Villaret, M. (2023). A good snowman is hard to plan. arXiv:2310.01471 [cs.AI].

Bonet, B., & Geffner, H. (1999). Planning as heuristic search: New results. In Biundo, S., & Fox, M. (Eds.), *Recent Advances in AI Planning. 5th European Conference on Planning (ECP 1999)*, Vol. 1809 of *Lecture Notes in Artificial Intelligence*, pp. 360–372, Heidelberg. Springer-Verlag.

Borgwardt, S., Hoffmann, J., Kovtunova, A., Krötzsch, M., Nebel, B., & Steinmetz, M. (2022). Expressivity of planning with horn description logic ontologies. In Honavar, V., & Spaan, M. (Eds.), *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, pp. 5503–5511. AAAI Press.

Brace, K. S., Rudell, R., & Bryant, R. E. (1990). Efficient implementation of a BDD package. In Smith, R. C. (Ed.), *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC 1990)*, pp. 40–45.

Bryant, R. E. (1985). Symbolic manipulation of Boolean functions using a graphical representation. In Ofek, H., & O'Neill, L. A. (Eds.), *Proceedings of the 22nd ACM/IEEE Conference on Design Automation (DAC 1985)*, pp. 688–694.

Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, *35*(8), 677–691.

Burch, J. R., Clarke, E. M., & Long, D. E. (1991). Symbolic model checking with partitioned transistion relations. In Halaas, A., & Denyer, P. B. (Eds.), *Proceedings of the International Conference on Very Large Scale Integration (VLSI 1991)*, pp. 49–58.

Burch, J. R., Clarke, E. M., Long, D. E., McMillan, K. L., & Dill, D. L. (1994). Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *13*(4), 401–424.

Christen, R., Eriksson, S., Katz, M., Muise, C., Petrov, A., Pommerening, F., Seipp, J., Sievers, S., & Speck, D. (2023). PARIS: Planning algorithms for reconfiguring independent sets. In Gal, K., Nowé, A., Nalepa, G. J., Fairstein, R., & Rădulescu, R. (Eds.), *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, pp. 453–460. IOS Press.

Ciardo, G., & Siminiceanu, R. (2002). Using edge-valued decision diagrams for symbolic generation of shortest paths. In Aagaard, M., & O'Leary, J. W. (Eds.), *Proceedings of the Fourth International Conference on Formal Methods in Computer-Aided Design (FMCAD 2002)*, Vol. 2517 of *Lecture Notes in Computer Science*, pp. 256–273. Springer-Verlag.

Cimatti, A., Pistore, M., Roveri, M., & Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, *147*, 35–84.

Corraya, S., Geißer, F., Speck, D., & Mattmüller, R. (2019). An empirical study of the usefulness of state-dependent action costs in planning. In Benzmüller, C., & Stuckenschmidt, H. (Eds.), *Proceedings of the 42nd Annual German Conference on Artificial Intelligence (KI 2019)*, Vol. 11793 of *Lecture Notes in Computer Science*, pp. 123–130. Springer-Verlag.

Cui, H., & Khardon, R. (2018). The SOGBOFA system in IPC 2018: Lifted BP for conformant approximation of stochastic planning. In *Sixth International Probabilistic Planning Competition (IPC-6): Planner Abstracts*, pp. 1–6.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Domshlak, C., Katz, M., & Shleyfman, A. (2015). Symmetry breaking in deterministic planning as forward search: Orbit space search algorithm. Tech. rep. IS/IE-2015-03, Technion.

Drechsler, R., & Becker, B. (1998a). *Binary Decision Diagrams – Theory and Implementation*. Springer.

Drechsler, R., & Becker, B. (1998b). Ordered Kronecker functional decision diagrams-a data structure for representation and manipulation of Boolean functions. *IEEE Trans. on CAD of Integrated Circuits and Systems*, *17*(10), 965–973.

Drexler, D., Gnad, D., Höft, P., Seipp, J., Speck, D., & Ståhlberg, S. (2023). Ragnarok. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Drexler, D., Seipp, J., & Speck, D. (2021). Subset-saturated transition cost partitioning. In Goldman, R. P., Biundo, S., & Katz, M. (Eds.), *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, pp. 131–139. AAAI Press.

Edelkamp, S. (2002). Symbolic pattern databases in heuristic search planning. In Ghallab, M., Hertzberg, J., & Traverso, P. (Eds.), *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, pp. 274–283. AAAI Press.

Edelkamp, S. (2003a). Limits and possibilities of PDDL for model checking software. In Edelkamp, S., & Hoffmann, J. (Eds.), *Proceedings of the ICAPS 2003 Workshop on the Competition: Impact, Organisation, Evaluation, Benchmarks*.

Edelkamp, S. (2003b). Promela planning. In Ball, T., & Rajamani, S. K. (Eds.), *Proceedings of the 10th International SPIN Workshop (SPIN 2003)*, Vol. 2648 of *Lecture Notes in Computer Science*, pp. 197–212. Springer-Verlag.

Edelkamp, S., & Helmert, M. (1999). Exhibiting knowledge in planning problems to minimize state encoding length. In Biundo, S., & Fox, M. (Eds.), *Recent Advances in AI Planning. 5th European Conference on Planning (ECP 1999)*, Vol. 1809 of *Lecture Notes in Artificial Intelligence*, pp. 135–147, Heidelberg. Springer-Verlag.

Edelkamp, S., & Helmert, M. (2000). On the implementation of MIPS. In Traverso, P., Veloso, M., & Giunchiglia, F. (Eds.), *Proceedings of the AIPS 2000 Workshop on Model-Theoretic Approaches to Planning*.

Edelkamp, S., & Helmert, M. (2001). The model checking integrated planning system (MIPS). *AI Magazine*, *22*(3), 67–71.

Edelkamp, S., & Kissmann, P. (2008). Limits and possibilities of BDDs in state space search. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pp. 1452–1453. AAAI Press.

Edelkamp, S., & Kissmann, P. (2009). Optimal symbolic planning with action costs and preferences. In Boutilier, C. (Ed.), *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pp. 1690–1695. AAAI Press.

Edelkamp, S., & Kissmann, P. (2011). On the complexity of BDDs for state space search: A case study in Connect Four. In Burgard, W., & Roth, D. (Eds.), *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pp. 18–23. AAAI Press.

Erol, K., Hendler, J. A., & Nau, D. S. (1996). Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence (AMAI)*, *18*(1), 69–93.

Fan, G., Müller, M., & Holte, R. (2017). The two-edged nature of diverse action costs. In Barbulescu, L., Frank, J., Mausam, & Smith, S. F. (Eds.), *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pp. 98–106. AAAI Press.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

Fišer, D., Torralba, Á., & Hoffmann, J. (2022a). Operator-potential heuristics for symbolic search. In Honavar, V., & Spaan, M. (Eds.), *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2022)*, pp. 9750–9757. AAAI Press.

Fišer, D., Torralba, Á., & Hoffmann, J. (2022b). Operator-potentials in symbolic search: From forward to bi-directional search. In Thiébaux, S., & Yeoh, W. (Eds.), *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pp. 80–89. AAAI Press.

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

Franco, S., Edelkamp, S., & Moraru, I. (2023). Complementarypdb. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Franco, S., Lelis, L. H. S., & Barley, M. (2018). The Complementary2 planner in the IPC 2018. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, pp. 32–36.

Franco, S., Torralba, Á., Lelis, L. H. S., & Barley, M. (2017). On creating complementary pattern databases. In Sierra, C. (Ed.), *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017)*, pp. 4302–4309. IJCAI.

Gazen, B. C., & Knoblock, C. A. (1997). Combining the expressivity of UCPOP with the efficiency of Graphplan. In Steel, S., & Alami, R. (Eds.), *Recent Advances in AI Planning. 4th European Conference on Planning (ECP 1997)*, Vol. 1348 of *Lecture Notes in Artificial Intelligence*, pp. 221–233. Springer-Verlag.

Geier, T., & Bercher, P. (2011). On the decidability of HTN planning with task insertion. In Walsh, T. (Ed.), *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pp. 1955–1961. AAAI Press.

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

Geißer, F., Keller, T., & Mattmüller, R. (2015). Delete relaxations for planning with state-dependent action costs. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pp. 1573–1579. AAAI Press.

Geißer, F., Keller, T., & Mattmüller, R. (2016). Abstractions for planning with state-dependent action costs. In Coles, A., Coles, A., Edelkamp, S., Magazzeni, D., & Sanner, S. (Eds.), *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2016)*, pp. 140–148. AAAI Press.

Geißer, F., & Speck, D. (2018). Prost-DD – Utilizing symbolic classical planning in THTS. In *Sixth International Probabilistic Planning Competition (IPC-6): Planner Abstracts*, pp. 13–16.

Geißer, F., Speck, D., & Keller, T. (2020). Trial-based heuristic tree search for MDPs with factored action spaces. In Harabor, D., & Vallati, M. (Eds.), *Proceedings of the 13th Annual Symposium on Combinatorial Search (SoCS 2020)*, pp. 38–47. AAAI Press.

Gerevini, A. E., Percassi, F., & Scala, E. (2024). An effective polynomial technique for compiling conditional effects away. In Dy, J., & Natarajan, S. (Eds.), *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, pp. 20104–20112. AAAI Press.

Ghosh, K., Dasgupta, P., & Ramesh, S. (2015). Automated planning as an early verification tool for distributed control. *Journal of Automated Reasoning*, *54*(1), 31–68.

Hansen, E. A., Zhou, R., & Feng, Z. (2002). Symbolic heuristic search using decision diagrams. In Koenig, S., & Holte, R. C. (Eds.), *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, Vol. 2371 of *Lecture Notes in Artificial Intelligence*, pp. 83–98. Springer-Verlag.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Haslum, P. (2011). Computing genome edit distances using domain-independent planning. In *ICAPS 2011 Scheduling and Planning Applications woRKshop*, pp. 45–51.

Haslum, P. (2013). Optimal delete-relaxed (and semi-relaxed) planning with conditional effects. In Rossi, F. (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pp. 2291–2297. AAAI Press.

Haslum, P., Ivankovic, F., Ramírez, M., Gordon, D., Thiébaux, S., Shivashankar, V., & Nau, D. S. (2018). Extending classical planning with state constraints: Heuristics and search for optimal planning. *Journal of Artificial Intelligence Research*, *62*, 373–431.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M. (2008). *Understanding Planning Tasks – Domain Complexity and Heuristic Decomposition*, Vol. 4929 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, *173*, 503–535.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?. In Gerevini, A., Howe, A., Cesta, A., & Refanidis, I. (Eds.), *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, pp. 162–169. AAAI Press.

Helmert, M., & Lasinger, H. (2010). The Scanalyzer domain: Greenhouse logistics as a planning problem. In Brafman, R., Geffner, H., Hoffmann, J., & Kautz, H. (Eds.), *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pp. 234–237. AAAI Press.

Hoffmann, J., & Edelkamp, S. (2005). The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, *24*, 519–579.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Holte, R. C. (2010). Common misconceptions concerning heuristic search. In Felner, A., & Sturtevant, N. (Eds.), *Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS 2010)*, pp. 46–51. AAAI Press.

Ivankovic, F., Gordon, D., & Haslum, P. (2019). Planning with global state constraints and state-dependent action costs. In Lipovetzky, N., Onaindia, E., & Smith, D. E. (Eds.), *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, pp. 232–236. AAAI Press.

Ivankovic, F., & Haslum, P. (2015). Optimal planning with axioms. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pp. 1580–1586. AAAI Press.

Jensen, R. M., Hansen, E. A., Richards, S., & Zhou, R. (2006). Memory-efficient symbolic heuristic search. In Long, D., Smith, S. F., Borrajo, D., & McCluskey, L. (Eds.), *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pp. 304–313. AAAI Press.

Karpas, E., & Magazzeni, D. (2020). Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, *3*, 417–439.

Katz, M., Sohrabi, S., Samulowitz, H., & Sievers, S. (2018). Delfi: Online planner selection for cost-optimal planning. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, pp. 57–64.

Kebschull, U., Schubert, E., & Rosenstiel, W. (1992). Multilevel logic synthesis based on functional decision diagrams. In Schweikert, D. G. (Ed.), *Proceedings of the 29th ACM/IEEE Design Automation Conference (DAC 1992)*, pp. 43–47.

Keller, T., & Eyerich, P. (2012). PROST: Probabilistic planning based on UCT. In McCluskey, L., Williams, B., Silva, J. R., & Bonet, B. (Eds.), *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, pp. 119–127. AAAI Press.

Keller, T., & Geißer, F. (2015). Better be lucky than good: Exceeding expectations in MDP evaluation. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, pp. 3540–3547. AAAI Press.

Keller, T., Pommerening, F., Seipp, J., Geißer, F., & Mattmüller, R. (2016). State-dependent cost partitionings for Cartesian abstractions in classical planning. In Kambhampati, S. (Ed.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pp. 3161–3169. AAAI Press.

Kissmann, P., & Edelkamp, S. (2009). Solving fully-observable non-deterministic planning problems via translation into a general game. In Mertsching, B., Hund, M., & Aziz, Z. (Eds.), *Proceedings of the 32nd Annual German Conference on Artificial Intelligence (KI 2009)*, Vol. 5803 of *Lecture Notes in Artificial Intelligence*, pp. 1–8. Springer-Verlag.

Kissmann, P., & Edelkamp, S. (2011). Improving cost-optimal domain-independent symbolic planning. In Burgard, W., & Roth, D. (Eds.), *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pp. 992–997. AAAI Press.

Kissmann, P., Edelkamp, S., & Hoffmann, J. (2014). Gamer and Dynamic-Gamer – Symbolic search at IPC 2014. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, pp. 77–84.

Kissmann, P., & Hoffmann, J. (2013). What's in it for my BDD? on causal graphs and variable orders in planning. In Borrajo, D., Kambhampati, S., Oddi, A., & Fratini, S. (Eds.), *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, pp. 327–331. AAAI Press.

Kissmann, P., & Hoffmann, J. (2014). BDD ordering heuristics for classical planning. *Journal of Artificial Intelligence Research*, *51*, 779–804.

Koehler, J., & Schuster, K. (2000). Elevator control as a planning problem. In Chien, S., Kambhampati, S., & Knoblock, C. A. (Eds.), *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, pp. 331–338. AAAI Press.

Koller, A., & Hoffmann, J. (2010). Waking up a sleeping rabbit: On natural-language sentence generation with FF. In Brafman, R., Geffner, H., Hoffmann, J., & Kautz, H. (Eds.), *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pp. 238–241. AAAI Press.

Kominis, F., & Geffner, H. (2015). Beliefs in multiagent planning: From one agent to many. In Brafman, R., Domshlak, C., Haslum, P., & Zilberstein, S. (Eds.), *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, pp. 147–155. AAAI Press.

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

Mattmüller, R., Geißer, F., Wright, B., & Nebel, B. (2018). On the relationship between state-dependent action costs and conditional effects in planning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, pp. 6237–6245. AAAI Press.

McDermott, D. (2000). The 1998 AI Planning Systems competition. *AI Magazine*, *21*(2), 35–55.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). PDDL – The Planning Domain Definition Language – Version 1.2. Tech. rep. CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers.

Minato, S. (1993). Zero-suppressed BDDs for set manipulation in combinatorial problems. In Dunlop, A. E. (Ed.), *Proceedings of the 30th Design Automation Conference (DAC 1993)*, pp. 272–277.

Miura, S., & Fukunaga, A. (2017). Automatic extraction of axioms for planning. In Barbulescu, L., Frank, J., Mausam, & Smith, S. F. (Eds.), *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pp. 218–227. AAAI Press.

Nebel, B. (2000). On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, *12*, 271–315.

Newell, A., & Simon, H. A. (1963). GPS: A program that simulates human thought. In Feigenbaum, E. A., & Feldman, J. (Eds.), *Computers and Thought*, pp. 279–293. Oldenbourg.

Nilsson, N. J. (1984). Shakey the robot. Tech. rep. 323, AI Center, SRI International. Menlo Park, CA, USA.

Palacios, H., & Geffner, H. (2009). Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, *35*, 623–675.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pednault, E. P. D. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R. J., Levesque, H. J., & Reiter, R. (Eds.), *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR 1989)*, pp. 324–332. Morgan Kaufmann.

Percassi, F., Scala, E., & Gerevini, A. E. (2024). Optimised variants of polynomial compilation for conditional effects in classical planning. In Felner, A., & Li, J. (Eds.), *Proceedings of the 17th Annual Symposium on Combinatorial Search (SoCS 2024)*, pp. 100–108. AAAI Press.

Pommerening, F., Helmert, M., Röger, G., & Seipp, J. (2015). From non-negative to general operator cost partitioning. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, pp. 3335–3341. AAAI Press.

Pommerening, F., Röger, G., & Helmert, M. (2013). Getting the most out of pattern databases for classical planning. In Rossi, F. (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pp. 2357–2364. AAAI Press.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Reffel, F., & Edelkamp, S. (1999). Error detection with directed symbolic model checking. In Wing, J. M., Woodcock, J., & Davies, J. (Eds.), *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM 1999)*, Vol. 1708 of *Lecture Notes in Computer Science*, pp. 195–211. Springer-Verlag.

Röger, G., Pommerening, F., & Helmert, M. (2014). Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In Schaub, T., Friedrich, G., & O'Sullivan, B. (Eds.), *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, pp. 765–770. IOS Press.

Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. In Lightner, M. R., & Jess, J. A. G. (Eds.), *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1993)*, pp. 42–47.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence — A Modern Approach*. Prentice Hall.

Sanner, S. (2010). Relational dynamic influence diagram language (RDDL): Language description..

Sanner, S., & McAllester, D. (2005). Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pp. 1384–1390. Professional Book Center.

Scholl, C., Möller, D., Molitor, P., & Drechsler, R. (1999). BDD minimization using symmetries. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *18*(2), 81–100.

Segovia-Aguas, J., Jiménez, S., & Jonsson, A. (2018). Computing hierarchical finite state controllers with classical planning. *Artificial Intelligence*, *62*, 755–797.

Seipp, J. (2018). Fast Downward Scorpion. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, pp. 77–79.

Seipp, J., & Helmert, M. (2018). Counterexample-guided Cartesian abstraction refinement for classical planning. *Journal of Artificial Intelligence Research*, *62*, 535–577.

Seipp, J., Pommerening, F., Sievers, S., & Helmert, M. (2017). Downward Lab. `https://doi.org/10.5281/zenodo.790461`.

Sievers, S., & Katz, M. (2018). Metis 2018. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, pp. 83–84.

Siminiceanu, R., & Roux, P. (2010). Model checking with edge-valued decision diagrams. In *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010)*, pp. 222–226.

Somenzi, F. (2015). CUDD: CU decision diagram package – release 3.0.0. https://github.com/ivmai/cudd. Accessed: 2023-09-19.

Speck, D. (2022). *Symbolic Search for Optimal Planning with Expressive Extensions*. Ph.D. thesis, University of Freiburg.

Speck, D. (2023). SymK – A versatile symbolic search planner. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Speck, D., Borukhson, D., Mattmüller, R., & Nebel, B. (2021). On the compilability and expressive power of state-dependent action costs. In Goldman, R. P., Biundo, S., & Katz, M. (Eds.), *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, pp. 358–366. AAAI Press.

Speck, D., Dornhege, C., & Burgard, W. (2017). Shakey 2016 – How much does it take to redo shakey the robot?. *IEEE Robotics and Automation Letters*, *2*(2), 1203–1209.

Speck, D., Geißer, F., & Mattmüller, R. (2018a). Symbolic planning with edge-valued multi-valued decision diagrams. In de Weerdt, M., Koenig, S., Röger, G., & Spaan, M. (Eds.), *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pp. 250–258. AAAI Press.

Speck, D., Geißer, F., & Mattmüller, R. (2018b). SYMPLE: Symbolic Planning based on EVMDDs. In *Ninth International Planning Competition (IPC-9): Planner Abstracts*, pp. 91–94.

Speck, D., Geißer, F., & Mattmüller, R. (2020). When perfect is not good enough: On the search behaviour of symbolic heuristic search. In Beck, J. C., Karpas, E., & Sohrabi, S. (Eds.), *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)*, pp. 263–271. AAAI Press.

Speck, D., Geißer, F., Mattmüller, R., & Torralba, Á. (2019). Symbolic planning with axioms. In Lipovetzky, N., Onaindia, E., & Smith, D. E. (Eds.), *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, pp. 464–472. AAAI Press.

Speck, D., Höft, P., Gnad, D., & Seipp, J. (2023). Finding matrix multiplication algorithms with classical planning. In Koenig, S., Stern, R., & Vallati, M. (Eds.), *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*, pp. 411–416. AAAI Press.

Speck, D., Mattmüller, R., & Nebel, B. (2020). Symbolic top-k planning. In Conitzer, V., & Sha, F. (Eds.), *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, pp. 9967–9974. AAAI Press.

Speck, D., Ortlieb, M., & Mattmüller, R. (2015). Necessary observations in nondeterministic planning. In Hölldobler, S., Krötzsch, M., Peñaloza-Nyssen, R., & Rudolph, S. (Eds.), *Proceedings of the 38th Annual German Conference on Artificial Intelligence (KI 2015)*, Vol. 9324 of *Lecture Notes in Artificial Intelligence*, pp. 181–193. Springer-Verlag.

Speck, D., Seipp, J., & Torralba, Á. (2024). Code, benchmarks and data for the paper "Symbolic Search for Cost-Optimal Planning with Expressive Model Extensions". `https://doi.org/10.5281/zenodo.12624111`.

Speicher, P., Steinmetz, M., Backes, M., Hoffmann, J., & Künnemann, R. (2018). Stackelberg planning: Towards effective leader-follower state space search. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, pp. 6286–6293. AAAI Press.

Taitler, A., Alford, R., Espasa, J., Behnke, G., Fišer, D., Gimelfarb, M., Pommerening, F., Sanner, S., Scala, E., Schreiber, D., Segovia-Aguas, J., & Seipp, J. (2024). The 2023 International Planning Competition. *AI Magazine*, *45*(2), 280–296.

Thiébaux, S., & Cordier, M.-O. (2001). Supply restoration in power distribution systems — A benchmark for planning under uncertainty. In Cesta, A., & Borrajo, D. (Eds.), *Proceedings of the Sixth European Conference on Planning (ECP 2001)*, pp. 196–202. AAAI Press.

Thiébaux, S., Hoffmann, J., & Nebel, B. (2005). In defense of PDDL axioms. *Artificial Intelligence*, *168*(1–2), 38–69.

Torralba, Á. (2015). *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. Ph.D. thesis, Universidad Carlos III de Madrid.

Torralba, Á. (2023). SymBD: A symbolic bidirectional search baseline. In *Tenth International Planning Competition (IPC-10): Planner Abstracts*.

Torralba, Á., Alcázar, V., Borrajo, D., Kissmann, P., & Edelkamp, S. (2014). SymBA*: A symbolic bidirectional A* planner. In *Eighth International Planning Competition (IPC-8): Planner Abstracts*, pp. 105–109.

Torralba, Á., Alcázar, V., Kissmann, P., & Edelkamp, S. (2017). Efficient symbolic search for cost-optimal planning. *Artificial Intelligence*, *242*, 52–79.

Torralba, Á., Linares López, C., & Borrajo, D. (2013). Symbolic merge-and-shrink for cost-optimal planning. In Rossi, F. (Ed.), *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pp. 2394–2400. AAAI Press.

Torralba, Á., Linares López, C., & Borrajo, D. (2018). Symbolic perimeter abstraction heuristics for cost-optimal planning. *Artificial Intelligence*, *259*, 1–31.

Torralba, Á., Speicher, P., Künnemann, R., Steinmetz, M., & Hoffmann, J. (2021). Faster Stackelberg planning via symbolic search and information sharing. In Leyton-Brown,

K., & Mausam (Eds.), *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, pp. 11998–12006. AAAI Press.

Vallati, M., Chrpa, L., Grześ, M., McCluskey, T. L., Roberts, M., & Sanner, S. (2015). The 2014 International Planning Competition: Progress and trends. *AI Magazine*, *36*(3), 90–98.

Yang, B., Bryant, R. E., O'Hallaron, D. R., Biere, A., Coudert, O., Janssen, G., Ranjan, R. K., & Somenzi, F. (1998). A performance study of BDD-based model checking. In Gopalakrishnan, G., & Windley, P. J. (Eds.), *Proceedings of the Second International Conference on Formal Methods in Computer-Aided Design (FMCAD 1998)*, pp. 255–289. Springer.

Yu, Z., Han, C., & Ma, Y. (2014). Emergency decision making: A dynamic approach. In Hiltz, S. R., Plotnick, L., Pfaf, M., & Shih, P. C. (Eds.), *Proceedings of the Eleventh International Conference on Information Systems for Crisis Response and Management (ISCRAM 2014)*, pp. 245–249. ISCRAM Association.