On Performance Guarantees for Symbolic Search in Classical Planning

David Speck and Malte Helmert

University of Basel, Switzerland
⟨davidjakob.speck, malte.helmert⟩@unibas.ch

Abstract. We show that standard symbolic search algorithms for classical planning can incur exponential overhead compared to explicit blind search in the presence of complex conditions and effects. To address this problem, we explore conjunctive partitioning in classical planning and present fully automated, domain-independent methods for representing actions and goal conditions in a partitioned form. We show that one of our methods, based on the Tseitin transformation, yields a symbolic search algorithm that in the worst case incurs only a polynomial overhead and in the best case can be exponentially more efficient than its explicit counterpart. Finally, our empirical evaluation shows that our theoretical findings carry over into practice: our algorithms solve planning problems previously intractable for symbolic search, and perform favorably overall compared to traditional symbolic search, explicit blind search, and other state-of-the-art planners.

1 Introduction

Symbolic search is an established method for classical cost-optimal planning. It is known to have complementary strengths to explicit heuristic search and is often assumed to be superior to explicit blind search. While this is often true for planning problems with simple conditions and effects, such as STRIPS [13] or SAS$^+$ [1], symbolic search algorithms face severe performance problems in the presence of complex conditions and effects [43]. This can be seen, for example, in the 2023 edition of the International Planning Competition [44]. In domains like Rubik's Cube, all planners based on symbolic search [37, 46, 17] were not able to solve a single task, although some of them were solvable with only a few actions, and explicit blind search easily solved those. The reason for this is that it is infeasible to represent the actions as logical formulas, as so-called transition relations, in the form of monolithic binary decision diagrams (BDDs) [2]. In this paper, we consider a conjunctive partitioned representation of these formulas to address this issue.

While the conjunctive partitioning of transition relations has long been standard in model checking [4], disjunctive partitioning has been considered the more appropriate approach in planning [47, 45]. However, this assumption is based on simple normalized forms of actions and goals. An exception is the work of Jensen and Veloso [22] on non-deterministic planning. It considers conjunctive partitioning of transition relations. However, it is based on a specialized modeling language tailored for this purpose called NADL. In addition, the preconditions of actions and the goal conditions are still represented as a monolithic BDD, which can lead to an exponentially large BDD representation and thus offers no performance guarantees compared to explicit blind search. There is a long history of studies on the performance guarantees of symbolic search and the sizes of the BDDs involved. Domain-specific analyses show that the state space or goal condition of planning problems, such as Gripper or the Connect-Four game, can be represented with polynomial or exponential time and size [9, 10]. Speck et al. [40] analyzed symbolic heuristic search [11] and showed that even the perfect heuristic can impose an exponential overhead over symbolic blind search. Using state-set branching [23], Fišer et al. [16, 14, 15] presented operator-potential heuristics for symbolic search with a concise heuristic representation that improves search but has no general performance guarantees compared to explicit or symbolic blind search. While all these approaches consider normalized goals and effects, Speck et al. [39, 43] investigated symbolic search for planning with axioms that allow the modeling of complex conditions. This approach relies on creating a monolithic BDD for each action that contains all complex effect conditions. If the formulas become too complex, this can lead to exponentially large BDD representations.

Contributions For decades, the state-of-the-art approach to planning as symbolic search has neglected conjunctive partitioning to represent goals and actions. On modern benchmark domains, however, this approach fails drastically due to the complexity of actions and the large representation size. In this paper, we present the first fully automated and domain-independent methods for the conjunctive representation of complex conditions and effects in planning. On the theoretical side, we propose a method that can effectively derive a variable-based conjunctive form [4] of complex actions. In contrast to previous work [22], this no longer requires manual modeling effort to partition the effects of actions. Furthermore, we propose an approach based on the Tseitin transformation [49] and show that it has a polynomial performance guarantee over explicit blind search. On the practical side, symbolic search with the proposed partitioned representations can for the first time effectively solve problems from domains with complex effects such as Rubik's Cube. This establishes a novel symbolic search approach that compares favorably to traditional symbolic search and other explicit search approaches.

2 Background

We provide the necessary background for our work.

2.1 Propositional Logic

A *literal* ℓ is either a Boolean variable v or its *negation* $\neg v$. A propositional *formula* ϕ is defined recursively as a truth value $\top$ (true) or

$\bot$ (false), a Boolean variable, the negation $\neg\phi$ of a formula ϕ, or the binary connection $(\phi \bullet \psi)$ of two formulas ϕ and ψ with a *conjunction* connector $(\phi \wedge \psi)$, *disjunction* connector $(\phi \vee \psi)$, *implication* connector $(\phi \rightarrow \psi)$, or *biimplication* connector $(\phi \leftrightarrow \psi)$. Implication $(\phi \rightarrow \psi)$ and biimplication $(\phi \leftrightarrow \psi)$ are equivalent to $\neg\phi \vee \psi$ and $(\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$. The *size* of a formula ϕ, i.e., the number of symbols, is denoted by $\|\phi\|$. With $\mathit{Vars}(\phi)$ we refer to the set of variables in a formula ϕ. With $\phi[v/\top]$ and $\phi[v/\bot]$ we refer to *restricting* a variable v in a formula ϕ to a particular value by replacing each occurrence of v in the formula with $\top$ or $\bot$. We write $\exists v\, \phi$ as a shorthand for $\phi[v/\top] \vee \phi[v/\bot]$, known as *forgetting* or *existentially quantifying* the variable v. Intuitively, by $\exists v\, \phi$ we remove the dependence of the formula ϕ on the variable v. When forgetting multiple variables $V = \{v_1, \ldots, v_n\}$ of a formula ϕ, we write $\exists V\, \phi$ as shorthand for $\exists v_1 \cdots \exists v_n\, \phi$. Renaming the variable v_1 to v_2 in the formula ϕ, written $\phi[v_1 \rightarrow v_2]$, means replacing all occurrences of v_1 to v_2 in ϕ. We require that v_2 does not already occur in ϕ before the renaming. For a set of variables $V = \{v_1, \ldots, v_n\}$ and a set of primed versions of those variables $V' = \{v'_1, \ldots, v'_n\}$, we write $\phi[V \rightarrow V']$ to rename all v_i to v'_i in ϕ, and $\phi[V' \rightarrow V]$ to rename all v'_i to v_i.

An *assignment* $s : V \mapsto \{\top, \bot\}$ maps each variable of V to $\top$ or $\bot$. Given an assignment s to the variables of a formula ϕ, the evaluation $\phi(s)$ yields $\top$ or $\bot$ for the formula using the usual rules of Boolean algebra [25]. If for an assignment s and a formula ϕ we have $\phi(s) = \top$, we say that s is a model for ϕ, written $s \models \phi$.

We use tuples to represent sequences of objects in any order. We write $\mathit{tup}(\{v_1, \ldots, v_n\}) = \langle v_1, \ldots, v_n \rangle$ for the *tuple* function, which transforms a set $\{v_1, \ldots v_n\}$ into a tuple in an arbitrary order. Also, $\oplus$ refers to the concatenation of two tuples, e.g., $\langle v_1 \rangle \oplus \langle v_2, v_3 \rangle = \langle v_1, v_2, v_3 \rangle$.

2.2 Conjunctive Normal Form

A formula is in *conjunctive normal form (CNF)* if it is a conjunction of one or more clauses, where a *clause* is a disjunction of literals. The naive approach to transform a formula ϕ into CNF applies standard laws of propositional logic [25], which however can lead to an exponential increase in the size of the formula [26]. We denote the naive transformation of a formula ϕ into CNF as $\mathit{cnf}(\phi)$.

An alternative CNF transformation is the Tseitin transformation [49]. To define it, we first introduce the notion of subformulas. If ϕ is a formula, the *immediate subformulas* of ϕ are defined as follows: 1) truth constants and variables have no immediate subformulas, 2) the only immediate subformula of $\phi = \neg\psi$ is ψ, and 3) the formula $\phi = \psi_1 \bullet \psi_2$ with $\bullet \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ has immediate subformulas ψ_1 and ψ_2. The set of *subformulas* of a formula ϕ is then defined as the smallest set $\mathit{Sub}(\phi)$ with $\phi \in \mathit{Sub}(\phi)$ and if $\psi \in \mathit{Sub}(\phi)$ then all immediate subformulas of ψ are in $\mathit{Sub}(\phi)$. The set of *proper subformulas* of ϕ is $\mathit{Sub}(\phi) \setminus \{\phi\}$.

Next, we define the Tseitin transformation [49, 26].

Definition 1 (Tseitin Transformation)**.** Let ϕ be a formula. The *Tseitin transformation* $\mathit{tsn}(\phi)$ is defined as

$$\mathit{tsn}(\phi) := [\phi] \wedge \bigwedge_{(\psi_1 \bullet \psi_2) \in \mathit{Sub}(\phi), \bullet \in \{\wedge,\vee,\rightarrow,\leftrightarrow\}} \mathit{cnf}([(\psi_1 \bullet \psi_2)] \leftrightarrow ([\psi_1] \bullet [\psi_2])) \wedge \bigwedge_{\neg\psi \in \mathit{Sub}(\phi), \neg\psi \text{ is not a literal}} \mathit{cnf}([\neg\psi] \leftrightarrow \neg[\psi])$$

$$\text{with} \quad [\chi] = \begin{cases} \chi & \text{if } \chi \text{ is a literal,} \\ v_\chi & \text{otherwise } (v_\chi \text{ is an auxiliary variable}) \end{cases}$$

V^{aux} denotes the set of *auxiliary variables* introduced. The clauses resulting from $\mathit{cnf}([(\psi_1 \bullet \psi_2)] \leftrightarrow ([\psi_1] \bullet [\psi_2]))$ *introduce* the auxiliary variable $v_{(\psi_1 \bullet \psi_2)}$, and the clauses of $\mathit{cnf}([\neg\psi] \leftrightarrow \neg[\psi])$ *introduce* the auxiliary variable $v_{\neg\psi}$. In addition, the clause $[\phi]$ *introduces* auxiliary variable v_ϕ if the input formula ϕ is not a literal.

For a formula ϕ, the Tseitin transformation produces a *quasi-equivalent* formula $\mathit{tsn}(\phi)$ in CNF. This means that there is a bijective mapping between the truth assignments of the original formula and the transformed one. A truth assignment for the transformed formula can be mapped to a truth assignment of the original formula by forgetting the auxiliary variables V^{aux}. Finally, the Tseitin transformation *tsn* can be generated in linear time and space in the size of the original formula [26].

Example 1. Consider a formula $\phi = (x \wedge y) \vee \neg z$ with subformulas $\mathit{Sub}(\phi) = \{(x \wedge y) \vee \neg z, (x \wedge y), \neg z, x, y, z\}$. We obtain $\mathit{tsn}(\phi) = v_{\phi_2} \wedge \mathit{cnf}(v_{\phi_2} \leftrightarrow (v_{\phi_1} \vee \neg z)) \wedge \mathit{cnf}(v_{\phi_1} \leftrightarrow (x \wedge y))$ where $\phi_2 = (x \wedge y) \vee \neg z$ and $\phi_1 = (x \wedge y)$. For $\mathit{cnf}(v_{\phi_1} \leftrightarrow (x \wedge y))$, we obtain the clauses $\mathcal{C}(v_{\phi_1}) = \{(\neg v_{\phi_1} \vee x), (\neg v_{\phi_1} \vee y), (v_{\phi_1} \vee \neg x \vee \neg y)\}$, which are the clauses that introduce variable v_{ϕ_1}. The remaining clauses $\mathcal{C}(v_{\phi_2}) = \{v_{\phi_2}, (\neg v_{\phi_2} \vee v_{\phi_1} \vee \neg z), (v_{\phi_2} \vee \neg v_{\phi_1}), (v_{\phi_2} \vee z)\}$ introduce v_{ϕ_2}.

2.3 Classical Planning

We consider grounded propositional planning tasks derived from a PDDL problem specification with full ADL features [30].

Definition 2 (Actions and States)**.** Let V be a finite set of Boolean *state variables*, and let all formulas be defined over V. An assignment s to V is called a *state*. An *action* a over V is a pair $\langle \mathit{pre}(a), \mathit{eff}(a) \rangle$, where $\mathit{pre}(a)$ is a formula describing the *precondition*, and $\mathit{eff}(a)$ is the *effect*, where effects are defined inductively as: 1) $\top$ is the *empty effect*, 2) v and $\neg v$ are *atomic effects* for $v \in V$, 3) $(e \wedge e')$ is a *conjunctive effect* if e, e' are effects, and 4) $(\phi \rhd e)$ is a *conditional effect* if ϕ is a formula and e is an effect. Each action a is associated with a *cost* $\mathit{cost}(a) \in \mathbb{N}_0$.

An action a has *size* $\|a\| = \|\mathit{pre}(a)\| + \|\mathit{eff}(a)\|$, where the size of its effect is defined inductively: the empty and atomic effects have size 1, a conjunctive effect has size equal to the sum of its components, and a conditional effect has size equal to the size of its condition formula plus that of its effect.

We define the semantics of an effect with the concept of *effcond* [32]. Intuitively, $\mathit{effcond}(\ell, e)$ represents the condition that must be true in the current state for the effect e to lead to the atomic effect ℓ.

Definition 3 (Effect Condition for an Effect)**.** Let ℓ be an atomic effect, and let e be an effect. The *effect condition* $\mathit{effcond}(\ell, e)$ under which ℓ triggers given the effect e is a formula defined as: 1) $\mathit{effcond}(\ell, \top) = \bot$, 2) $\mathit{effcond}(\ell, e) = \top$ for the atomic effect $e = \ell$, 3) $\mathit{effcond}(\ell, e) = \bot$ for all atomic effects $e = \ell' \neq \ell$, 4) $\mathit{effcond}(\ell, (e \wedge e')) = (\mathit{effcond}(\ell, e) \vee \mathit{effcond}(\ell, e'))$, and 5) $\mathit{effcond}(\ell, (\phi \rhd e)) = (\phi \wedge \mathit{effcond}(\ell, e))$.

The application of an action with its effects to a state is then defined as follows.

Definition 4 (Application of Actions and Effects)**.** The *resulting state* $s' = s[\![e]\!]$ of applying e in s is defined as follows for all $v \in V$:

$$s'(v) = \begin{cases} \top & \text{if } s \models \mathit{effcond}(v, e) \\ \bot & \text{if } s \models \mathit{effcond}(\neg v, e) \wedge \neg\mathit{effcond}(v, e) \\ s(v) & \text{otherwise} \end{cases}$$

Action a is *applicable* in s if $s \models \mathit{pre}(a)$. If a is applicable in s, the *resulting state* of *applying* a in s, written $s[\![a]\!]$, is the state $s[\![\mathit{eff}(a)]\!]$.

We define a planning task as follows.

Definition 5 (Planning Task)**.** A *planning task* $\Pi = \langle V, I, A, \gamma \rangle$ consists of a finite set of propositional *state variables* V, an assignment I of V called the *initial state*, a finite set of *actions* A over V and a *goal formula* γ over V.

The objective of classical planning is to find a plan. A *plan* $\pi = \langle a_1, \ldots, a_n \rangle$ for planning task Π is a sequence of applicable actions that generates a sequence of states $s_0, \ldots, s_n$, where $s_0 = I$, $s_n \models \gamma$, and $s_i = s_{i-1}[\![a_i]\!]$ for all $i = 1, \ldots, n$. The cost of a plan $\pi = \langle a_1, \ldots, a_n \rangle$ is defined as the sum of the costs of its actions, i.e., $\sum_{i=1}^{n} \mathit{cost}(a_i)$. A plan is *optimal* if no other plan has a lower cost.

Let v be a variable and a an action. We define $\mathit{regr}(v, \mathit{eff}(a))$ as a shorthand for $\mathit{effcond}(v, \mathit{eff}(a)) \vee (v \wedge \neg \mathit{effcond}(\neg v, \mathit{eff}(a)))$. Intuitively, $\mathit{regr}(v, a)$ describes the condition under which v is true after applying the effect of a. This concept is known as regression [32].

Example 2. Consider two variables x, y and an action $a = \langle \neg x, (\neg y \rhd x) \wedge y \rangle$. For a state $s = \{x \mapsto \bot, y \mapsto \bot\}$, action a is applicable since $s \models \mathit{pre}(a)$. For literal x, we get $\mathit{effcond}(x, \mathit{eff}(a)) = \mathit{effcond}(x, (\neg y \rhd x) \wedge y) = \mathit{effcond}(x, \neg y \rhd x) \vee \mathit{effcond}(x, y) = (\neg y \wedge \mathit{effcond}(x, x)) \vee \bot = \neg y \wedge \top = \neg y$. Further, we get $\mathit{effcond}(y, \mathit{eff}(a)) = \top$ and $\mathit{effcond}(\neg x, \mathit{eff}(a)) = \mathit{effcond}(\neg y, \mathit{eff}(a)) = \bot$. Thus, applying a in s yields state $s[\![a]\!] = \{x \mapsto \top, y \mapsto \top\}$.

2.4 Binary Decision Diagrams

Binary decision diagrams are data structures that can be used to represent propositional formulas [27].

Definition 6 (Binary Decision Diagram)**.** Let V be a set of Boolean variables. A *binary decision diagram (BDD)* B is a directed acyclic graph with a single root node and two terminal nodes: the $\bot$-sink and the $\top$-sink. Each inner node is labeled with a variable $v \in V$ and has two successors connected by labeled edges: the *low edge*, representing that variable v is false, and the *high edge*, representing that variable v is true. The *size* $|B|$ of a BDD B is the number of its nodes.

We consider BDDs in a reduced and ordered form [2]. In planning, BDDs are often used to represent formulas and sets of states. A BDD B represents the set of states S where $s \in S$ iff traversing the BDD according to s leads to the $\top$-sink. We consider the previously defined logical operations $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\exists$ (forgetting), $[v_1 \to v_2]$ (variable renaming) for BDDs, which yield the corresponding BDD representing the modified formula [2, 29]. By $\mathit{bdd}(\phi)$ we refer to the BDD representing formula ϕ that results from a bottom-up creation of ϕ.

In this paper, we use multiple sets of variables V, V' and V^{aux} for BDDs. V refers to the (unprimed) state variables of a planning task, V' refers to a primed copy of such state variables and V^{aux} are the auxiliary variables introduced by the Tseitin transformation. Importantly, if a formula ϕ does not refer to certain variables, the BDD representation of ϕ also does not refer to those variables, i.e., has no decision nodes on those variables. Thus, we freely introduce new variables for some BDDs, which does not affect the size of BDDs that do not refer to such variables.

The order of the variables has a significant impact on the size of a BDD. For certain formulas ϕ, there is an exponential difference in the size of $\mathit{bdd}(\phi)$ depending on the variable order. Most importantly for this work, there are formulas that have exponentially large BDDs with respect to the size of the formula independently of the order of the variables. One such formula is the *connect-two* formula which, for a grid of size $n \times n$, describes all vertically or horizontally adjacent cells [10]. We assume that the primed and unprimed versions of the variables alternate in the variable order. The auxiliary variables are positioned after all others.

2.5 Symbolic and Explicit Search

We describe *symbolic search* for cost-optimal classical planning [29, 48]. To start the search, all relevant components of a planning task — initial state, goal, and actions — are transformed into logical formulas and represented as BDDs. Then, symbolic search is split into two phases, a reachability phase and a plan reconstruction phase.

The reachability phase maintains a *closed* and *open* list. The *closed* list stores already expanded states, while the *open* list contains states that have not yet been considered. Both lists consist of multiple BDDs list_g parameterized by cost g, where closed_g contains all expanded states and open_g all states reachable with cost g. In each step, the BDD open_g representing the states with the lowest cost g is extracted from *open*. Then the function *eval-goal* is used to check whether open_g contains a goal state, usually implemented as the intersection of open_g and the BDD representing the goal formula. If the intersection is non-empty, a goal state has been found and the plan reconstruction is triggered to return the corresponding optimal plan (see below). Otherwise, the *image* operation [e.g., 29, 48] is applied to compute all successor states of open_g with respect to the actions of the planning task. The successors of an action a with cost $\mathit{cost}(a)$ are then filtered by removing all already expanded states stored in *closed* and added to the BDD $\mathit{open}_{c+\mathit{cost}(a)}$. Finally, the algorithm terminates either when a goal state (and thus a plan) is found or when *open* becomes empty, in which case the task is unsolvable.

In symbolic search, the parent state that caused a state to be generated is not directly known, so a plan reconstruction is necessary once a goal state is found. This is possible by performing a greedy search in the reverse direction, using the optimal path costs stored in *closed* as perfect heuristic estimates. More precisely, starting from the found goal state s with cost g, the predecessors of s with respect to each action a are computed using the *preimage* operation, until a predecessor is found in $\mathit{closed}_{g-\mathit{cost}(a)}$. Once such a predecessor s' of s is found, the same procedure is repeated with s' until the initial state is reached, and the actual plan induced by this trace can be returned (omitting some details in the presence of zero-cost actions).

While the above description of symbolic search is sufficient to follow this paper, we refer the interested reader to more elaborate explanations in Torralba [45] and Speck et al. [43].

By *explicit (blind) search* we refer to uniform-cost search, i.e., Dijkstra's algorithm [6]. We sometimes refer to explicit search as the explicit counterpart of symbolic search because the main difference is that symbolic search processes complete sets of states at once while explicit search handles states individually. For explicit search, we consider a straightforward implementation, i.e., all states and formulas are processed iteratively. In particular, we assume that no sophisticated duplicate detection or successor generation techniques are used [e.g., 18]. This is a reasonable assumption, since these techniques usually require special normal forms of actions.

Both symbolic search and explicit search are *complete* and *optimal*, meaning that for a given planning task they return a plan if one exists and that plan is optimal.

3 Representing Formulas of a Planning Task

In this section, we derive how planning tasks with complex conditions and effects can be efficiently handled as logical formulas. First, we define a generalized version of the transition relation for complex actions. Then, we present and discuss three methods for representing goal formulas and transition relations with varying degrees of conjunctive partitioning for symbolic search: one is the standard approach used in planning, one is adapted from model checking (used here for the first time for domain-independent planning), and the third is a novel method based on the Tseitin transformation. Finally, we analyze the theoretical properties of these representations, establish performance guarantees for symbolic search using the proposed representations, and conclude with a set of optimizations.

3.1 Complex Actions as Formulas

While the goal of a planning task is already given as a logical formula, symbolic search also requires that each action a is represented as a logical formula $\tau(a)$. Previous approaches to classical planning as symbolic search assume that actions are in a normalized form, such as SAS$^+$ [48] or FDR [19, 24]. Thus, we introduce the following general definition of a transition relation for actions, directly derived from PDDL problem specifications with all ADL features [30], eliminating the need for prior normalization, which can be computationally expensive.

Definition 7 (Transition Relation)**.** Let a be an action, V a set of state variables, and V' a primed copy of the set of state variables. The *transition relation* of a is defined as

$$\tau(a) := \mathit{pre}(a) \wedge \bigwedge_{v \in V} \tau(v, a),$$

with $\tau(v, a) = (\mathit{regr}(v, \mathit{eff}(a)) \leftrightarrow v')$.

The first conjunct of the transition relation encodes the precondition. The big conjunction relates the new values of the variables $v \in V$, represented by $v' \in V'$, to the old state described by the variables V. More precisely, a variable v' is true iff $\mathit{regr}(v, \mathit{eff}(a))$ holds, which exactly encodes the condition under which v is true after applying action a. Intuitively, the transition relation $\tau(a)$ of an action a describes all state pairs, represented as predecessor (over V) and successor states (over V'), such that applying a in one of the predecessors is allowed and yields one of the successors.

Lemma 1. *Transforming an action a into a logical formula $\tau(a)$ is bounded by $O(|V| \cdot \|a\|)$ in time and space.*

Proof. The precondition $\mathit{pre}(a)$ is bounded by $\|a\|$. Transforming $\mathit{effcond}(v, \mathit{eff}(a))$ into a logical formula is asymptotically linear in time and space in the size of the effect $\|e\|$, which itself is bounded by the action size $\|a\|$. This is because *effcond* traverses each subeffect (atomic effect, conjunction effect, etc.) of effect e exactly once, recursively applying the rules defined in Def. 3. Since the size of e depends on the subeffects of e, the computation of *effcond* is linear in $\|e\|$. We have $\mathit{regr}(v, \mathit{eff}(a)) = \mathit{effcond}(v, \mathit{eff}(a)) \vee (v \wedge \neg \mathit{effcond}(\neg v, \mathit{eff}(a)))$ for every variable in V. Thus, the formula $\tau(a)$ is bounded by $O(\|a\| + 2 \cdot |V| \cdot \|e\|) = O(|V| \cdot \|a\|)$. □

Importantly, the size of an action $\|a\|$ can be greater than the number of variables $|V|$. For example, if action a has conditional effects on each variable in V, then the time and space to construct $\tau(a)$ can grow quadratically with the number of variables, i.e., $\Omega(|V|^2)$.

Algorithm 1: Central functions of symbolic search.

```
1  function formula-as-bdds(φ : formula)
2      return ⟨bdd(C) | C ∈ cf(φ)⟩
3  function image(states : bdd, T_a : bdds)
4      succ ← states
5      foreach t ∈ T_a do
6          succ ← succ ∧ t
7      return (∃(V ∪ V^aux) succ)[V' → V]
8  function eval-goal(states : bdd, G : bdds)
9      goal ← states
10     foreach g ∈ G do
11         goal ← goal ∧ g
12     return ∃V^aux goal
13 function cpreimage(s : bdd, T_a : bdds, closed : bdd)
14     state-pairs ← s[V → V'] ∧ closed
15     foreach t ∈ T_a do
16         state-pairs ← state-pairs ∧ t
17     return ∃(V' ∪ V^aux) state-pairs
```

3.2 Conjunctive Representations

For symbolic search we must represent all relevant formulas of a planning task as BDDs. The critical components are the actions (as transition relations) and the goal formula, while the initial state is negligible, as it is just a conjunction of atoms. We *conjunctively partition* these formulas: the tuple of formulas $\langle \phi_1, \ldots, \phi_n \rangle$ represents the formula $\phi = \phi_1 \wedge \cdots \wedge \phi_n$. The implementation represents each formula ϕ_i as a BDD and the partitioned formula ϕ as a tuple of BDDs (function *formula-as-bdds* in Alg. 1). More precisely, *formula-as-bdds* returns a tuple of BDDs whose conjunction represents the function ϕ, where *cf* specifies how the formula is partitioned. Note that the other functions in Alg. 1 are introduced later in the paper.

3.2.1 Action-Monolithic Representation

Let us consider the approach common in automated planning of representing each action and the goal formula using a single monolithic BDD within symbolic search [e.g., 8, 48].

Definition 8 (Action-Monolithic Form)**.** Let ϕ be a formula, we define the *action-monolithic form* cf_{A} as $cf_{\mathrm{A}}(\phi) = \langle \phi \rangle$.

In the action-monolithic form, we create with *formula-as-bdds* (Alg. 1) a single monolithic BDD for each action and for the goal formula. This is not to be confused with having a single monolithic transition relation for all actions by disjunctive merging [7, 47]. Note that allowing rich actions with complex preconditions and effects (Def. 7) goes beyond the state of the art in symbolic planning, which considers only normalized forms with worst-case exponential preprocessing [e.g., 48, 43].

3.2.2 Variable-Monolithic Representation

The next representation for complex actions uses one conjunct per effect variable. This approach was originally described for model checking [4]. In model checking, the dynamics of the world are typically described in a variable-centric way, meaning that transitions are specified independently for each variable. This is also the reason why

Burch et al. [4] introduced the modeling language NADL to describe nondeterministic planning tasks, which allows the direct specification of the transition of each variable and thus naturally supports a conjunctive partitioned form. In contrast, classical planning tasks are typically modeled in an action-centric manner using PDDL [28], and actions are often assumed to have a simple structure. As a result, a conjunctive representation has historically not been considered natural in planning [45], and it was not immediately clear how to derive it. Based on our definition of the transition relation (Def. 7), we can automatically transform classical planning actions into the conjunctive form originally proposed by Burch et al. [4]. Note that the preconditions and the goal are still represented as a single monolithic BDD.

Definition 9 (Variable-Monolithic Form)**.** Let $\Pi = \langle V, I, A, \gamma \rangle$ be a planning task with $V = \{v_1, \ldots, v_n\}$. We define the *variable-monolithic form* cf_{V} of either the goal formula γ or the transition relations $\tau(a)$ with $a \in A$ as

$$cf_{\text{V}}(\phi) = \begin{cases} \langle \gamma \rangle & \text{if } \phi = \gamma, \\ \langle pre(a), \tau(v_1, a), \ldots, \tau(v_n, a) \rangle & \text{if } \phi = \tau(a). \end{cases}$$

In the variable-monolithic form, the symbolic planning algorithm represents the goal formula as a tuple with a single BDD, while the transition relation of each action is represented as a tuple with $1+|V|$ BDDs, one for the precondition and one for each variable. This neatly aligns with the structure of our transition relation definition (Def. 7), which consists of the precondition and a conjunction of $\tau(v, a)$ over each variable $v \in V$, which are explicitly kept separate here.

3.2.3 Tseitin Representation

The final approach to representing actions and goal formulas of a planning task is based on the Tseitin transformation [49]. The idea is to bring potentially complex transition relations and goal formulas into a CNF without imposing a significant increase in the size of the representation. We call this novel representation of formulas of a planning task the *Tseitin form*. It will be key to guaranteeing that symbolic search can incur only a polynomial worst-case overhead over explicit blind search, while still being exponentially more efficient in the best case. The main reason is that even the transition function for a single variable can be highly complex, leading to an infeasibly large representation for the previously considered forms.

Definition 10 (Tseitin Form)**.** For a formula ϕ, let $\langle \phi_1, \ldots, \phi_n \rangle$ be an order of its subformulas $Sub(\phi)$ without literals such that for all ϕ_i and ϕ_j, if ϕ_i is a proper subformula of ϕ_j, then $i < j$. We define the *Tseitin form* cf_{T} as

$$cf_{\text{T}}(\phi) = \begin{cases} \langle tsn(\phi) \rangle & \text{if } tsn(\phi) \text{ is a literal,} \\ tup(\mathcal{C}(v_{\phi_1})) \oplus \cdots \oplus tup(\mathcal{C}(v_{\phi_n})) & \text{otherwise,} \end{cases}$$

where $\mathcal{C}(v_{\phi_i})$ is the set of clauses of $tsn(\phi)$ that introduce the auxiliary variable v_{ϕ_i}.

Intuitively, for a formula ϕ, the Tseitin form represents each clause of the CNF formula $tsn(\phi)$ as a separate BDD using *formula-as-bdds* (Alg. 1). In addition, we impose an order on the clauses induced by the partial order of the subformulas of ϕ. More precisely, we have all clauses corresponding to a formula ϕ_i before the formula ϕ_j if ϕ_i is a proper subformula of ϕ_j. The idea behind this is that we only introduce a new auxiliary variable v_{ϕ_j} if all auxiliary variables v_{ϕ_i} on which v_{ϕ_j} depends have already been fully introduced. So if we build the sequential conjunction with a state set S, we also sequentially fix the values of the auxiliary variables for each state $s \in S$.

Example 3. Consider the formula $(x \wedge y) \vee \neg z$ from Example 1. With the Tseitin transformation we get the clause sets $\mathcal{C}(v_{\phi_1})$ and $\mathcal{C}(v_{\phi_2})$, which introduce the two auxiliary variables $v_{\phi_1} = v_{(x \wedge y)}$ and $v_{\phi_2} = v_{(x \wedge y) \vee \neg z}$. Since ϕ_1 is a proper subformula of ϕ_2, we get the order $\langle \phi_1, \phi_2 \rangle$. So the Tseitin form is the tuple $tup(\mathcal{C}(v_{\phi_1})) \oplus tup(\mathcal{C}(v_{\phi_2}))$.

4 Symbolic Search for Complex Actions and Goals

We now describe how symbolic search can be implemented using the different representations aimed at handling complex actions and goals by explaining the important operations of symbolic search, which are outlined as functions in Alg. 1. In particular, for all three representations we describe the successor computation (*image*), the goal checking (*eval-goal*), and the constrained predecessor computation necessary for plan reconstruction (*cpreimage*). The handling of auxiliary variables and constrained predecessor computation within symbolic search are essential for the novel performance guarantees derived later.

Image (Alg. 1, lines 3–7) computes all successor states *succ* of a given set of states *states* for a transition relation representing action a. In the action-monolithic case, we compute the standard *image* operation on BDDs, $(\exists V\, (states \wedge T_a))[V' \rightarrow V]$ [4, 48]. For the variable-monolithic and Tseitin forms, we use the partitioned form of a transition relation T_a. Consider a transition relation $T_a = t_1 \wedge t_2 \wedge t_3$ and the for loop of the *image* function of Alg. 1. In the variable-monolithic and Tseitin cases we compute $(((states \wedge t_1) \wedge t_2) \wedge t_3)$, while in the action-monolithic case we have the computation $(states \wedge T_a) = (states \wedge (t_1 \wedge t_2 \wedge t_3))$. This difference in computation is the most critical part of the partitioned representation. Instead of computing the complete predecessor-successor relation of an action as a monolithic BDD, we reorder the conjunction to place the state BDD *states* first. This allows us to compute only those successors that are relevant with respect to *states*. Finally, *image* computes equivalent formulas in all three cases. Compared to the action-monolithic form, the variable-monolithic form uses a different conjunction order to yield an equivalent formula (associative law). In Tseitin form, a quasi-equivalent formula is computed which is equivalent after forgetting auxiliary variables [49, 26].

Eval-goal (Alg. 1, lines 8–12) returns all goal states of a given set of states *states*. In the action- and variable-monolithic form, this simply computes $states \wedge bdd(\gamma)$. In the Tseitin form, we compute a quasi-equivalent formula using a conjunctive partitioned representation of G. After forgetting all auxiliary variables, this is equivalent to the formula obtained with a monolithic goal representation.

Cpreimage (Alg. 1, lines 13–17) is defined for plan reconstruction to identify predecessors of states already visited during forward reachability and stored in the closed list *closed*. Thus, we define the *cpreimage* to compute the predecessors of a state s contained in *closed* with respect to an action a. In the action-monolithic case, we compute the formula $\exists V'\, (s[V \rightarrow V'] \wedge closed \wedge T_a)$. *cpreimage* is almost identical to *preimage* used traditionally for plan reconstruction [e.g., 45, 43], but it moves the intersection with the *closed* states into the predecessor computation, which is traditionally performed after computing the predecessor states. Intuitively, this adds the formula representing *closed* to the transition relation as an additional precondition. We do this because the set of predecessors and its BDD representation can be huge for complex actions, and we are only interested in the subset *closed*. With *cpreimage* we avoid representing this set of states first and then computing the intersection with the desired states in *closed*. For the partitioned forms of T_a, with the same

argument as for the *image* computation, we get equivalent formulas as a result of *cpreimage* with all three representations. Plan reconstruction is typically not considered in model checking, where the emphasis is usually on showing that states with certain conditions are unreachable or on evaluating general LTL formulas [31, 5].

4.1 Theoretical Properties

We start by showing that symbolic search with our proposed representations is complete and optimal.

Theorem 1. *Symbolic search using the* action-monolithic*,* variable-monolithic*, and* Tseitin *representations is* complete *and* optimal*.*

Proof. Symbolic search is complete and optimal with standard definitions of *image*, *cpreimage*, and *eval-goal*. For the action-monolithic form, all tuples of BDDs are singletons, and the functions are equivalent to the standard definitions. For the other forms, it is therefore sufficient to show that the resulting formulas are equivalent to those in the action-monolithic case. In the variable-monolithic case, the formulas produced are equivalent, since they differ only in the order of the conjunctions (associative law). For the Tseitin form, each function computes a quasi-equivalent formula before forgetting auxiliary variables. By forgetting these variables, we obtain an equivalent formula. □

It can be shown that symbolic search using any of the three representations can be exponentially more efficient than explicit blind search. Consider the Gripper domain, where a robot with two grippers has to transport n balls between two rooms. The state space grows exponentially with the number of balls, so explicit blind search requires exponential time and memory in the input size [20]. However, it is known that any instance of the Gripper domain can be solved in polynomial time and memory relative to the encoding size using standard symbolic search [9, 45]. Since Gripper is a STRIPS domain, where the preconditions and effects of actions as well as the goal formula are conjunctions of literals, the following applies.

Proposition 1. *For some inputs, explicit search can require* exponentially more memory and time *than symbolic search using the* action-monolithic*,* variable-monolithic*, or* Tseitin *representations.*

Next, we show that symbolic search with the action- and variable-monolithic representations can be exponentially worse than explicit search. This holds even if an early goal test is performed and in the presence of a concisely representable goal.

Theorem 2. *For some inputs, symbolic search using the* action-monolithic *or* variable-monolithic *representations can require* exponentially more memory and time *than explicit search, regardless of the variable order.*

Proof. Consider a family of planning tasks where the variables represent a grid. We have a single action a that has the connect-two formula [10] as a precondition, and all tasks have the unique plan $\langle a \rangle$. In explicit blind search, the precondition of a must be evaluated once for the initial state, which can be done in polynomial time and space. However, generating the action- or variable-monolithic BDD representation of a needs exponential time and memory in $\sqrt{\|a\|}$ (see Lemma 4 in Edelkamp and Kissmann [10]). □

We show that symbolic search with the Tseitin form has performance guarantees relative to explicit search, requiring at most a polynomial increase in time and memory in the size of the input planning task. While this overhead may still affect practical performance, it is preferable to the exponential overhead of previous approaches.

Theorem 3. *For all inputs, symbolic search using the* Tseitin representation *can require at most* $O(|V|\cdot\|\gamma\|+|V|^2\cdot\|a_{\max}\|)$ *more memory and time than explicit search, where* $\|a_{\max}\| = \max_{a\in A}\|a\|$.

Proof sketch (Full proof in Appendix). The size of a BDD representing states S is bounded by $O(|S|\cdot|V|)$ [12]. With Tseitin, we get $O(\|\phi\|)$ clauses of constant size for a formula ϕ. For *eval-goal*, we iterate over the clauses of *tsn*(γ), building a BDD representing a fixed number of states over $V \cup V^{\text{aux}}$. The intermediate BDDs have size $O(|S|\cdot|V\cup V^{\text{aux}}|) = O(|S|\cdot|V|\cdot\|\gamma\|)$, since V^{aux} are auxiliary variables of *tsn*(γ). *eval-goal* has $O(\|\gamma\|)$ steps (#clauses), resulting in a complexity of $O(|S|\cdot|V|\cdot\|\gamma\|^2)$. For *image* with action a, we can reason analogously: the transition relation has size $\tau(a) \in O(|V|\cdot\|a\|)$, leading to $O(|S|\cdot|V|^2\cdot\|a\|^2)$. In comparison, explicit search has time and memory bounds $\Omega(|S|\cdot\|\gamma\|)$ for goal evaluation and $\Omega(|S|\cdot\|a\|)$ for successor generation with action a. □

4.2 Optimizations

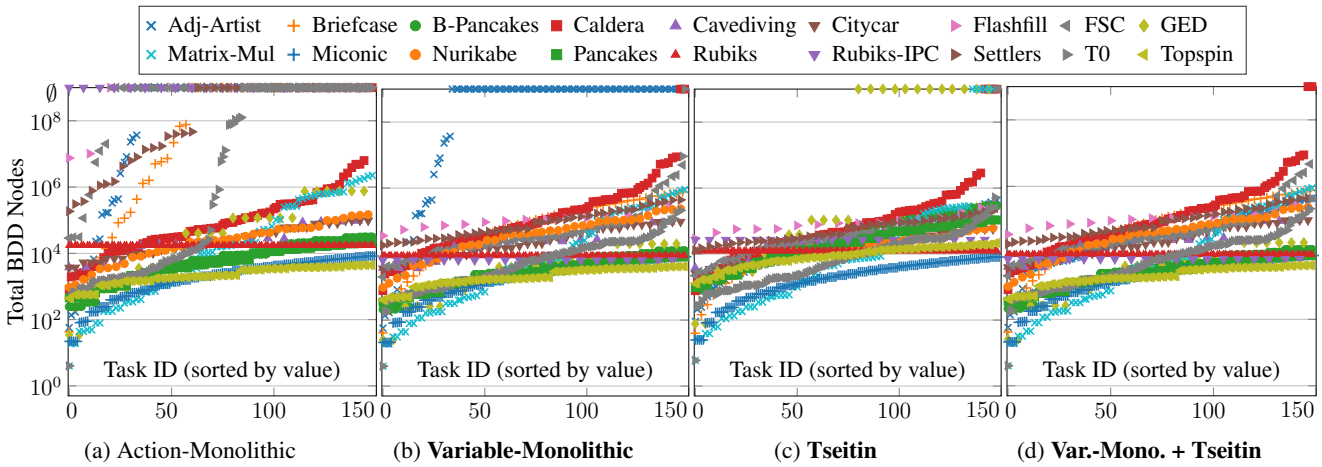
Several performance optimizations can be applied directly to symbolic search with partitioned representations of formulas. Early quantification can be performed on a conjunctive partitioned transition relation, allowing immediate forgetting of variables not referenced in later conjuncts [4]. Partitioned BDDs can be merged with time and size constraints, often improving runtime at the expense of memory. Auxiliary variables that appear in only one conjunct can be safely forgotten. Transition relations can be combined by disjunction once an action-monolithic form is obtained [48]. Finally, backward and bidirectional search can be realized using *image* for successor and *cpreimage* (without constraints) for predecessor computation.

5 Experiments

We implemented the presented symbolic search (including the optimizations) in the SymK planner [41], which is based on Fast Downward 23.06 [18]. We evaluate forward, backward, and bidirectional search with and without transition relation merging, limited to 100k nodes. We use CUDD [36] as the BDD library, the Gamer variable order [24], and an optimized Tseitin transformation introducing a single auxiliary variable for conjunctions of multiple literals. In addition to our three representations, we evaluate a fourth variant, which tries for five seconds to build a formula in the variable-monolithic form and then in the Tseitin form.

We compare our approaches to explicit blind search and the top non-portfolio planner in the 2023 International Planning Competition (IPC), Scorpion [34], which implements an explicit A* search with a saturated cost partitioning heuristic [35]. For each run, we use a time and memory limit of 30 min and 6 GiB.

We consider domains with complex conditional effects, including official IPC domains from all classical tracks [21, 33, 42, 3, 44]. The *Adjacent Artist* domain is newly created to stress test our approaches. It comprises a grid of different sizes and contains the connect-two formula as an effect condition. We only consider tasks that can be grounded within resource limits. Note that during grounding, preconditions and goal formulas are normalized to conjunctions of literals [19]. In the future, we would like to investigate other grounding approaches that may benefit our novel symbolic search approach. Our code and experimental data are available online [38].



(a) Action-Monolithic (b) **Variable-Monolithic** (c) **Tseitin** (d) **Var.-Mono. + Tseitin**

Figure 1: Sizes of BDDs (unique nodes) required to represent planning tasks with complex effects along the y-axis. Tasks that cannot be represented as BDDs within the resource limits are marked with ∅. Newly introduced representations for classical planning are shown in bold.

	Expl.		Action-Mono.						**Variable-Mono.**						**Tseitin**						**Var.-Mono. + Tseitin**					
Domain	Blind	Scorp.	Fwd	Bwd	Bid	Fwd M	Bwd M	Bid M	Fwd	Bwd	Bid	Fwd M	Bwd M	Bid M	Fwd	Bwd	Bid	Fwd M	Bwd M	Bid M	Fwd	Bwd	Bid	Fwd M	Bwd M	Bid M
Adj-Artist (100)	8	8	23	11	23	23	16	23	23	11	23	23	16	23	51	7	51	**90**	13	**90**	51	11	51	**90**	15	**90**
Briefcase (50)	7	**13**	8	6	9	8	6	9	8	6	9	8	6	9	8	6	8	8	6	9	8	6	9	8	6	9
B-Pancakes (100)	35	38	30	31	48	31	35	**49**	30	31	47	31	35	**49**	26	26	32	31	35	**49**	30	31	46	31	35	**49**
Caldera (78)	21	26	25	8	23	**29**	9	25	24	8	23	**29**	8	25	23	8	23	**29**	8	25	24	8	23	**29**	8	25
Cavediving (17)	**4**	**4**	**4**	0	**4**	**4**	0	**4**	**4**	0	**4**	**4**	0	**4**	**4**	0	**4**	**4**	0	**4**	**4**	0	**4**	**4**	0	**4**
Citycar (40)	10	17	**19**	0	13	**19**	0	18	**19**	0	13	**19**	0	18	18	0	13	**19**	0	18	**19**	0	13	**19**	0	18
Flashfill (15)	0	0	0	0	0	2	0	2	1	0	1	**6**	0	**6**	1	0	1	**6**	0	**6**	1	0	1	**6**	0	**6**
FSC (57)	19	19	7	0	6	7	0	6	19	0	17	**20**	0	**20**	12	0	11	**20**	0	19	19	0	17	**20**	0	**20**
GED (26)	**20**	**20**	**20**	8	14	**20**	8	**20**	**20**	8	11	**20**	8	**20**	14	8	10	14	8	14	**20**	8	10	**20**	8	**20**
Matrix-Mul (77)	22	30	30	30	33	30	30	**39**	30	30	33	30	30	**39**	30	30	33	30	30	**39**	30	30	33	30	30	**39**
Miconic (150)	79	147	124	108	125	**150**	149	**150**	124	108	123	**150**	149	**150**	123	109	124	**150**	149	**150**	123	108	124	**150**	149	**150**
Nurikabe (38)	16	**20**	13	4	12	16	7	16	13	4	12	16	7	16	13	4	12	16	7	16	13	4	12	16	7	16
Pancakes (100)	37	39	35	35	**52**	35	38	**52**	34	35	51	35	38	**52**	26	26	37	35	38	**52**	33	35	51	35	38	**52**
Rubiks (100)	35	43	25	25	**50**	25	25	**50**	25	25	45	25	25	**50**	19	15	30	25	25	**50**	25	25	45	25	25	**50**
Rubiks-IPC (20)	8	**10**	0	0	0	0	0	0	4	1	4	5	1	6	4	1	4	5	1	6	4	1	4	5	1	6
Settlers (40)	8	**10**	8	0	9	9	0	9	7	0	6	9	0	9	4	0	5	9	0	9	7	0	7	9	0	9
T0 (119)	27	38	35	23	35	40	21	39	39	23	38	**44**	21	43	37	23	37	**44**	21	**44**	39	23	39	**44**	21	**44**
Topspin (100)	23	31	26	27	45	26	27	**46**	24	25	43	26	27	**46**	18	18	29	26	27	45	24	25	43	26	27	**46**
Σ (1227)	379	513	432	316	501	474	371	557	448	315	503	500	371	585	431	281	464	561	368	645	474	315	532	567	370	**653**
Norm. Σ (18)	5.5	6.9	5.5	3.2	6.0	6.0	3.7	6.9	6.0	3.3	6.2	6.8	3.7	7.7	5.5	2.9	5.7	7.3	3.7	8.2	6.2	3.3	6.4	7.5	3.7	**8.4**

Table 1: Coverage, i.e., the number of solved tasks, for explicit search and symbolic search with multiple action representations, search directions (forward, backward, bidirectional), and transition relation merging (M). Newly introduced representations for classical planning are shown in bold. The sum (Σ) denotes the total number of solved tasks, while the normalized sum (Norm. Σ) denotes the aggregated percentage of solved instances per domain.

Representation Size Figure 1 shows the cumulative size of BDDs required to represent the initial state, goal, and transition relations for each task of 18 domains. We see that the action-monolithic form fails to create these data structures in multiple domains, among them the IPC 2023 Rubik's Cube domain. The variable-monolithic and Tseitin representations perform much better, but the variable-monolithic form cannot represent the connect-two formula of the Adjacent Artist domain. The Tseitin form has trouble representing the larger instances of the Matrix Multiplication and GED domains. In these domains the action formulas become huge, sometimes with hundreds of thousands of auxiliary variables, causing memory issues with the BDD library. Our hybrid of the variable-monolithic and Tseitin forms combines the best of all approaches for these domains.

Coverage Table 1 shows a domain-wise comparison of the number of tasks solved by each search algorithm. The results show that the merging of the partitioned transition relations (M) is crucial for the performance, as also found for the action-monolithic case [48]. In practice, the Tseitin form without merging suffers in certain domains from the polynomial overhead compared to explicit search and the large number of conjuncts compared to the other symbolic forms. However, in domains like Adjacent Artist or Rubik's Cube, the partitioned forms pay off over the action-monolithic form. With merged transition relations, the Tseitin form shows a clear advantage over explicit search. Overall, the new representations outperform the action-monolithic form, especially when all optimizations are applied. By combining the variable-monolithic and Tseitin forms with transition-relation merging and bidirectional search, we introduce a novel symbolic search approach that outperforms the state of the art in both symbolic and explicit search, solving 140 more tasks than Scorpion, the top non-portfolio planner at IPC 2023 [44].

6 Conclusions

We presented new methods to advance symbolic search for classical planning using conjunctive partitioning. In contrast to previous work, we propose a fully automated and domain-independent approach to partition the relevant BDDs representing actions and goal conditions conjunctively. For the Tseitin method, we established a performance guarantee relative to explicit search: in the worst case, it requires only polynomially more time and memory than its explicit counterpart, whereas all previous approaches may incur exponential overhead. Our empirical evaluation shows that our new representations of actions and goals not only improve symbolic search but also allow it to surpass modern explicit search approaches, solving significantly more tasks on the considered benchmarks with conditional effects than the state of the art.

As future work, we aim to conduct similar investigations for symbolic search with heuristics [11, 16], with the goal of establishing performance guarantees relative to explicit heuristic search.

Appendix: Full Proof of Theorem 3

Theorem 3. *For all inputs, symbolic search using the* Tseitin representation *can require at most $O(|V|\cdot\|\gamma\|+|V|^2\cdot\|a_{\max}\|)$ more memory and time than explicit search, where $\|a_{\max}\| = \max_{a\in A}\|a\|$.*

Proof. In symbolic search with BDDs, the critical operations compared to explicit search are successor generation (*image*), goal evaluation (*eval-goal*), and plan reconstruction. Other tasks, such as adding or removing states from the open or closed lists, are performed iteratively in explicit search, while symbolic search handles sets of states in bulk. However, the worst-case complexity remains the same, since BDDs can in the worst case degenerate into representations equivalent to explicit state tables, since the size of a BDD representing a set of states S over variables V is bounded by $O(|S|\cdot|V|)$ [12]. Furthermore, forgetting and renaming operations on a BDD representing states S over variables V can be performed in linear time and memory with respect to the represented states and variables, i.e., in $O(|S|\cdot|V|)$, since in the worst case we can consider an explicit state table.

eval-goal: We consider an input BDD *states* representing state set S. With the Tseitin transformation, we obtain $O(\|\gamma\|)$ clauses of constant size (at most three literals) for the transformed goal formula γ. For *eval-goal*, we iterate over the clauses of *tsn*(γ), building a BDD that represents a fixed number of states $|S|$ over $V\cup V^{\text{aux}}$. This holds because each conjunction in the for-loop may introduce new auxiliary variables from V^{aux}, but due to the clause order, their values are immediately fixed, preserving the number of represented assignments $|S|$ over $V\cup V^{\text{aux}}$. Furthermore, the intermediate BDDs have size $O(|S|\cdot|V\cup V^{\text{aux}}|) = O(|S|\cdot(|V|+\|\gamma\|))$, since V^{aux} are auxiliary variables introduced by *tsn*(γ) and hence linear in $\|\gamma\|$. Now we can derive that $O(|S|\cdot(|V|+\|\gamma\|)) = O(S|\cdot|V|+|S|\cdot\|\gamma\|)$ is upper bounded by $O(|S|\cdot|V|\cdot\|\gamma\|)$, because $O(|S|\cdot|V|)$ and $O(|S|\cdot\|\gamma\|)$ is bounded by $O(|S|\cdot|V|\cdot\|\gamma\|)$. The for-loop of *eval-goal* has $O(\|\gamma\|)$ steps (one per clause), and each step performs a conjunction with a constant-sized BDD, so the total time and memory complexity is $O(\|\gamma\|\cdot(|S|\cdot|V|\cdot\|\gamma\|)) = O(|S|\cdot|V|\cdot\|\gamma\|^2)$.

image: We consider an input BDD *states* representing the state set S. For the successor generation we can reason analogously to *eval-goal*. However, there are two important differences: the transition relation $\tau(a)$ of an action a has size $\tau(a)\in O(|V|\cdot\|a\|)$ (Lemma 1), so we get $O(|V|\cdot\|a\|)$ clauses, and with the conjunctions we introduce new variables not only over V^{aux} but over $V'\cup V^{\text{aux}}$. For the number of assignments in the intermediate BDDs, we get the bound of $O(|S|\cdot|V\cup V'\cup V^{\text{aux}}|) = O(|S|\cdot(2|V|+\|\tau(a)\|))$, since $|V| = |V'|$ and $|V^{\text{aux}}|$ is bounded by $O(\|\tau(a)\|)$. Continuing the transformation: $O(|S|\cdot(2|V|+\|\tau(a)\|)) = O(|S|\cdot(|V|+\|\tau(a)\|)) = O(|S|\cdot(|V|+|V|\cdot\|a\|)) = O(|S|\cdot|V|\cdot\|a\|)$. Since there are $O(|V|\cdot\|a\|)$ clauses, and with the for-loop each corresponds to a conjunction of a constant-sized BDD with a BDD of size $O(|S|\cdot|V|\cdot\|a\|)$, we derive a complexity of $O(|S|\cdot|V|^2\cdot\|a\|^2)$.

get-plan: The function *get-plan* is executed once and reverses the successor calculation of *image*. Since we only consider predecessor-successor pairs that we already considered in the forward direction with *image*, the time and memory consumption of the reachability analysis performed before can at most double, which is only a constant factor that becomes negligible asymptotically.

In the explicit variant of symbolic search, for a state set S, the time and memory complexity for goal evaluation is $\Omega(|S|\cdot\|\gamma\|)$, and for successor generation with an action a, it is $\Omega(|S|\cdot\|a\|)$, since each state and formula must be processed individually. Compared to symbolic search, this results in an overhead of $O(|V|\cdot\|\gamma\|)$ in time and memory for goal evaluation and $O(|V|^2\cdot\|a_{\max}\|)$ for successor generation, where $a_{\max}$ is the action with the largest representation size. □

Acknowledgements

This work 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

[1] C. Bäckström and B. Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

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

[3] C. Büchner, P. Ferber, J. Seipp, and M. Helmert. Abstraction heuristics for factored tasks. In *Proc. ICAPS 2024*, pages 40–49, 2024.

[4] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Proc. VLSI 1991*, pages 49–58, 1991.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[7] S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *Proc. ECP 1999*, pages 135–147, 1999.

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

[9] S. Edelkamp and P. Kissmann. Limits and possibilities of BDDs in state space search. In *Proc. AAAI 2008*, pages 1452–1453, 2008.

[10] S. Edelkamp and P. Kissmann. On the complexity of BDDs for state space search: A case study in Connect Four. In *Proc. AAAI 2011*, pages 18–23, 2011.

[11] S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *Proc. KI 1998*, pages 81–92, 1998.

[12] S. Eriksson, G. Röger, and M. Helmert. Unsolvability certificates for classical planning. In *Proc. ICAPS 2017*, pages 88–97, 2017.

[13] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *AIJ*, 2:189–208, 1971.

[14] D. Fišer, Á. Torralba, and J. Hoffmann. Operator-potential heuristics for symbolic search. In *Proc. AAAI 2022*, pages 9750–9757, 2022.

[15] D. Fišer, Á. Torralba, and J. Hoffmann. Operator-potentials in symbolic search: From forward to bi-directional search. In *Proc. ICAPS 2022*, pages 80–89, 2022.

[16] D. Fišer, Á. Torralba, and J. Hoffmann. Boosting optimal symbolic planning: Operator-potential heuristics. *AIJ*, 334:104174, 2024.

[17] S. Franco, S. Edelkamp, and I. Moraru. ComplementaryPDB Planner. In *IPC-10 Planner Abstracts*, 2023.

[18] M. Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

[19] M. Helmert. Concise finite-domain representations for PDDL planning tasks. *AIJ*, 173:503–535, 2009.

[20] M. Helmert and G. Röger. How good is almost perfect? In *Proc. AAAI 2008*, pages 944–949, 2008.

[21] J. Hoffmann and S. Edelkamp. The deterministic part of IPC-4: An overview. *JAIR*, 24:519–579, 2005.

[22] R. M. Jensen and M. M. Veloso. OBDD-based universal planning: Specifying and solving planning problems for synchronized agents in non-deterministic domains. In M. Wooldridge and M. M. Veloso, editors, *Artificial Intelligence Today*, volume 1600 of *LNCS*, pages 213–248. Springer-Verlag, 1999.

[23] R. M. Jensen, M. M. Veloso, and R. E. Bryant. State-set branching: Leveraging BDDs for heuristic search. *AIJ*, 172(2–3):103–139, 2008.

[24] P. Kissmann, S. Edelkamp, and J. Hoffmann. Gamer and Dynamic-Gamer – Symbolic search at IPC 2014. In *IPC-8 Planner Abstracts*, pages 77–84, 2014.

[25] H. Kleine Büning and T. Lettmann. *Propositional Logic: Deduction and Algorithms*, volume 48 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1999.

[26] E. Kuiter, S. Krieter, C. Sundermann, T. Thüm, and G. Saake. Tseitin or not Tseitin? The impact of CNF transformations on feature-model analyses. In *Proc. ASE 2022*, pages 110:1–110:13, 2022.

[27] C.-Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

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

[29] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[30] E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. KR 1989*, pages 324–332, 1989.

[31] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 46–57, 1977.

[32] J. Rintanen. Regression for classical and nondeterministic planning. In *Proc. ECAI 2008*, pages 568–572, 2008.

[33] J. Segovia-Aguas, S. Jiménez, and A. Jonsson. Computing hierarchical finite state controllers with classical planning. *JAIR*, 62:755–797, 2018.

[34] J. Seipp. Scorpion 2023. In *IPC-10 Planner Abstracts*, 2023.

[35] J. Seipp, T. Keller, and M. Helmert. Saturated cost partitioning for optimal classical planning. *JAIR*, 67:129–167, 2020.

[36] F. Somenzi. CUDD: CU decision diagram package – Release 3.0.0. https://github.com/cuddorg/cudd, 2015. Accessed July 31, 2025.

[37] D. Speck. SymK – A versatile symbolic search planner. In *IPC-10 Planner Abstracts*, 2023.

[38] D. Speck and M. Helmert. Code, benchmarks and data for the ECAI 2025 paper "On Performance Guarantees for Symbolic Search in Classical Planning". https://doi.org/10.5281/zenodo.16640080, 2025.

[39] D. Speck, F. Geißer, R. Mattmüller, and Á. Torralba. Symbolic planning with axioms. In *Proc. ICAPS 2019*, pages 464–472, 2019.

[40] D. Speck, F. Geißer, and R. Mattmüller. When perfect is not good enough: On the search behaviour of symbolic heuristic search. In *Proc. ICAPS 2020*, pages 263–271, 2020.

[41] D. Speck, R. Mattmüller, and B. Nebel. Symbolic top-k planning. In *Proc. AAAI 2020*, pages 9967–9974, 2020.

[42] D. Speck, P. Höft, D. Gnad, and J. Seipp. Finding matrix multiplication algorithms with classical planning. In *Proc. ICAPS 2023*, pages 411–416, 2023.

[43] D. Speck, J. Seipp, and Á. Torralba. Symbolic search for cost-optimal planning with expressive model extensions. *JAIR*, 82:1349–1405, 2025.

[44] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fišer, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, J. Segovia-Aguas, and J. Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296, 2024. doi: 10.1002/aaai.12169.

[45] Á. Torralba. *Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning*. PhD thesis, Universidad Carlos III de Madrid, 2015.

[46] Á. Torralba. SymBD: A symbolic bidirectional search baseline. In *IPC-10 Planner Abstracts*, 2023.

[47] Á. Torralba, S. Edelkamp, and P. Kissmann. Transition trees for cost-optimal symbolic planning. In *Proc. ICAPS 2013*, pages 206–214, 2013.

[48] Á. Torralba, V. Alcázar, P. Kissmann, and S. Edelkamp. Efficient symbolic search for cost-optimal planning. *AIJ*, 242:52–79, 2017.

[49] G. Tseitin. On the complexity of derivation in the propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York, 1968. English Translation.