Computing Planning Centroids and Minimum Covering States Using Symbolic Bidirectional Search

Alberto Pozanco[1], **Álvaro Torralba**[2], **Daniel Borrajo**[1*]
[1]J.P. Morgan AI Research
[2]Aalborg University, Aalborg, Denmark
{alberto.pozancolancho,daniel.borrajo}@jpmorgan.com, alto@cs.aau.dk

Abstract

In some scenarios, planning agents might be interested in reaching states that keep certain relationships with respect to a set of goals. Recently, two of these types of states were proposed: centroids, which minimize the average distance to the goals; and minimum covering states, which minimize the maximum distance to the goals. Previous approaches compute these states by searching forward either in the original or a reformulated task. In this paper, we propose several algorithms that use symbolic bidirectional search to efficiently compute centroids and minimum covering states. Experimental results in existing and novel benchmarks show that our algorithms scale much better than previous approaches, establishing a new state-of-the-art technique for this problem.

Introduction

Automated Planning typically deals with the task of finding a sequence of actions, namely a plan, which achieves a goal state from a given initial state (Ghallab, Nau, and Traverso 2004). However, in some scenarios, planning agents might be interested in reaching states that keep certain relationships with respect to a set of (potential) goals. Recently, two of these types of states were proposed (Pozanco et al. 2019; Karpas 2022): centroids, which minimize the average distance (cost) to the goals; and minimum covering states, which minimize the maximum distance to the goals. Finding states with these properties has proven to be useful in diverse settings. For example, they were used in deceptive planning (Price et al. 2023), where an agent seeks a plan such that observers are not able to discern its goal. There are also potential uses in anticipatory planning (Burns et al. 2012), where agents start acting before a goal arrives. For example, agents that reply to incoming requests from customers should not be idle when no request is active, but instead proactively gather information that might be needed to reply to most of the customers' potential future inquiries.

Figure 1 illustrates these states in the planning task introduced by Pozanco et al. (2019). The ranger should generate a plan to set the camp at a location that minimizes the cost (time) of a plan to put out any fire that might break out.

[*]On leave from Univ. Carlos III de Madrid

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

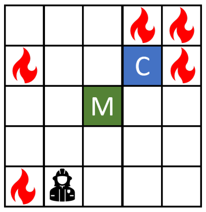


Figure 1: Centroid (C, blue) and minimum covering state (M, green) of a forest ranger planning task. Flames mark goal locations where the ranger might need to put out fires.

There exist two approaches in the literature to compute these states. The first one (Pozanco et al. 2019) uses a best-first search algorithm to explore the state space of the original planning task, computing the distance to each goal from each state, and returning the state that optimizes the given metric. This is a naïve approach, as it needs to (i) explore all the reachable states; and (ii) compute an optimal plan to each goal from each state in order to get the perfect distance. The second approach (Karpas 2022) compiles the original planning task into a larger classical planning task with multiple agents, one for each goal. They proposed two slightly different compilations for centroids and minimum covering states inspired on seminal work on goal recognition design (Keren, Gal, and Karpas 2014). Empirically, Karpas (2022) showed that computing centroid and minimum covering states by solving the reformulated tasks was orders of magnitude faster than using the exhaustive search approach by Pozanco et al. (2019) in most tasks. However, this approach still presents some drawbacks. First, different compilations must be developed for each optimization criteria, so it is difficult to generalize the approach beyond centroids and minimum covering states. Second, compiled tasks will typically yield larger state spaces (potentially involving 0-cost actions) that will often result in harder tasks that are more difficult to be efficiently solved by planners. Finally, compiling all goals into a single problem makes the exploitation of their independence more challenging.

In this paper we propose to use symbolic bidirectional search to compute centroids, minimum covering states, and more broadly any state that optimizes a given multi-goal distance function. Our family of Symbolic Multi-Goal

(SMG) algorithms offers several benefits over previous approaches. First, unlike (Karpas 2022), we perform search in the original task rather than in a reformulated one. Second, while (Pozanco et al. 2019) needs to optimally solve a large number of planning tasks to get the perfect distance to each goal from each state, we can compute this distance by searching backwards independently from each goal. Once we find a candidate state, we can check if it is reachable from the initial state by searching forward with that state as goal, removing the need of exhausting the state space. Finally, symbolic search uses succinct data structures to represent and manipulate sets of states, which is an efficient alternative when exhaustive search is needed.

The rest of the paper is organized as follows. We first formalize classical planning and the foundations of our approach: symbolic and bidirectional search. Then, we extend the standard centroid and minimum covering states definitions to states that optimize a given multi-goal distance function. Next, we introduce SMG, a family of symbolic bidirectional search algorithms to compute such states. Finally, we evaluate our new algorithms in existing and novel benchmarks, showing that they outperform previous approaches in computing planning centroid and minimum covering states.

Preliminaries

A SAS$^+$ planning task (Bäckström and Nebel 1995) is a tuple $\Pi = \langle V, I, \mathcal{O}, G \rangle$. V is a finite set of state variables, each associated with a finite domain D_v. A partial state p is a function on a subset of variables $V_p \subseteq V$ that assigns each variable $v \in V_p$ a value in its domain, $p[v]$. A state s is a complete assignment to all the variables. With $\mathcal{S}$ we refer to the set of all possible states defined over V. We also use partial states to represent conditions on states. A state s satisfies a condition p ($s \models p$) if $s(v) = p(v)$ for all $v \in V_p$. We also identify any partial state p with the set of states that satisfy it: $S_p = \{s \mid s \models p\}$. The state $I \in \mathcal{S}$ is the initial state of the planning task, and G is the goal condition, which defines the set of goal states $\mathcal{S}_G \subseteq \mathcal{S}$.

$\mathcal{O}$ is a set of operators, where an operator is a tuple $o = \langle pre_o, \mathit{eff}_o, c_o \rangle$ of partial variable assignments called preconditions and effects, respectively, and $c_o \in \mathbb{N}_0$ is the non-negative cost of o. An operator $o \in \mathcal{O}$ is applicable in state s iff $s \models pre_o$. Applying operator o in state s results in a state $s[o]$ where $s[o](v) = \mathit{eff}_o(v)$ for all $v \in V_{\mathit{eff}_o}$ and $s[o](v) = s(v)$ for all other variables.

A sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ is applicable in a state s_0 if there are states $s_1, \ldots, s_n$ such that o_i is applicable in s_{i-1} and $s_i = s_{i-1}[o_i]$ for all $i = 1, \ldots, n$. The resulting state of this application is $s_0[\pi] = s_n$, and $c(\pi) = \sum_{o_i \in \pi} c_{o_i}$ denotes the cost of π. A state s is reachable iff there exists a sequence of operators π applicable in I such that $I[\pi] = s$. With $\mathcal{S}_R \subseteq \mathcal{S}$ we refer to the set of all reachable states of the planning task. The solution to a planning task Π is a plan, i.e., a sequence of operators π such that $I[\pi] \in \mathcal{S}_G$. A plan with minimal cost is optimal.

We denote as $h^*(s, s')$ the optimal cost of reaching state s' from state s. If there is no path between the two states, $h^*(s, s') = \infty$. We denote as $g(s) = h^*(I, s)$, and $h^*_G(s) : \mathcal{S} \mapsto \mathbb{N}_0 \cup \{\infty\}$ for a goal G as $\min_{s_G \in S_G} h^*(s, s_G)$.

Symbolic Representation with Decision Diagrams

Binary Decision Diagrams (BDDs) (Bryant 1986) are a efficient data-structure to encode Boolean functions $\{0,1\}^n \mapsto \{\top, \bot\}$. We use BDDs to represent sets of states $S \subseteq \mathcal{S}$. This requires some arbitrary encoding of the values of the state variables V in binary. To simplify the presentation, we assume without loss of generality that the set of variables V have a binary domain. Each set of states S is represented by a BDD $\mathcal{B}$ encoding its characteristic function $\mathcal{X}_S : \mathcal{S} \mapsto \{\top, \bot\}$, where $\mathcal{X}_S(s) = \top$ iff $s \in S$. Slightly abusing notation, we will write $s \in \mathcal{B}$ whenever s is in the set of states represented by $\mathcal{B}$.

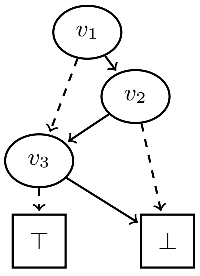
A BDD (Figure 2a) is a directed acyclic graph with a single root node and up to two terminal nodes, $\top$ and $\bot$. Each inner node corresponds to a binary variable $v \in V$, and has two successors depending on whether $s(v) = 0$ (dashed edge), or $s(v) = 1$ (solid edge). Given a state $s \in \mathcal{S}$, and a BDD $\mathcal{B}$, it can be checked whether $s \in \mathcal{B}$ by a simple top-down traversal, which will always end in $\top$ if $s \in \mathcal{B}$ and $\bot$ otherwise. For example, the BDD in Figure 2a has 5 nodes and represents a set with 3 states: $\{v_1 \mapsto 0, v_2 \mapsto 0, v_3 \mapsto 0\}$, $\{v_1 \mapsto 0, v_2 \mapsto 1, v_3 \mapsto 0\}$, and $\{v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 0\}$. We denote as $\mathcal{B}_p$ the BDD representing a (partial) state p, i.e., the set of states $S_p = \{s \mid s \models p\}$. The size of a BDD $|\mathcal{B}|$ is the number of nodes in its directed acyclic graph. In general, the number of states represented by a BDD can be exponentially larger than its number of nodes.

We assume BDDs are *reduced* and *ordered*. BDDs are ordered whenever variables are always checked in the same order (though some variables may be skipped) in any path from the root to the leaves. BDDs are reduced whenever (a) there are no irrelevant nodes in which both successors point to the same node; and (b) all nodes are unique (i.e., any equivalent nodes with the same variable and successors are merged). These properties are easy to maintain. Also, they allow the implementation of efficient operations on BDDs, whose runtime depends on the size of the BDDs and not on how many states they represent (which, again, could be exponentially larger). Specifically, the union ($\cup$) and intersection ($\cap$) of sets of states can be computed as the disjunction ($\mathcal{B} \vee \mathcal{B}'$) and conjunction ($\mathcal{B} \wedge \mathcal{B}'$) of their characteristic functions, respectively. The runtime of these operations is $O(|\mathcal{B}||\mathcal{B}'|)$. However, the conjunction/disjunction of n BDDs is worst-case exponential in n.

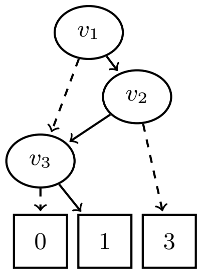
Algebraic Decision Diagrams (ADDs) (Bahar et al. 1997) are similar to BDDs, but have an arbitrary number of terminal nodes with different discrete values (see Figure 2b). That is, we can use them to represent functions mapping states to numerical values $\mathcal{S} \mapsto \mathbb{N}_0 \cup \{\infty\}$.

The APPLY operation takes as input two ADDs and a binary operation $\circ$ (e.g. $+, -, \times, \div, \max, \min$), and returns an ADD $\mathcal{A} = \text{APPLY}(\mathcal{A}_1, \mathcal{A}_2, \circ)$ representing the function $\mathcal{A}(s) = \mathcal{A}_1(s) \circ \mathcal{A}_2(s)$. Thus, using apply, we define any binary operation over ADDs, e.g, we write $\mathcal{A}_1 + \mathcal{A}_2$ as a shorthand for $\text{APPLY}(\mathcal{A}_1, \mathcal{A}_2, +)$. The runtime is $O(|\mathcal{A}_1||\mathcal{A}_2|)$.

ADDs can be converted into BDDs and vice versa (Torralba 2015; Speck 2022). Given an ADD $\mathcal{A}$, and a value x, the function $\text{BDD}(\mathcal{A}, x)$ returns a BDD representing the set



(a) BDD



(b) ADD

Figure 2: Example of a BDD and an ADD.

of states $\{s \mid \mathcal{A}(s) = x\}$. For example, in Figure 2, the BDD corresponds to $\text{BDD}(\mathcal{A}, 0)$. Given a BDD $\mathcal{B}$ and two values x, y, the function $\text{ADD}(\mathcal{B}, x, y)$ returns an ADD $\mathcal{A}$ where $\mathcal{A}(s) = x$ if $s \in \mathcal{B}$ and $\mathcal{A}(s) = y$ otherwise. Also, for any ADD $\mathcal{A}$, it is possible to compute the value $\min_s \mathcal{A}(s)$, as that corresponds to the terminal node in $\mathcal{A}$ with the lowest value. Similarly, we can sample any state minimizing the function, by obtaining a path from the root to such terminal node. All of these operations can be done in a single traversal so they have running time linear in the size of the BDD/ADD.

Symbolic Search

Symbolic search explores the state space by using the representation of sets of states as BDDs explained above. To do search directly in the symbolic representation, operators are represented as transition relations (TRs). A set of operators $O \subseteq \mathcal{O}$ can be represented as a TR containing the set of all state pairs (s, s') such that s' is reachable from s by applying an operator $o \in O$. For a given set of states $\mathcal{B}$ and TR T, the image/preimage operator computes all successors/predecessors of $\mathcal{B}$ with respect to the operators represented by T.

In the forward direction, the search starts with a BDD representing the initial state $\mathcal{B}_I$, and iteratively constructs a set of BDDs with an associated cost $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \ldots$, where $\mathcal{B}_g = \{s \mid h^*(I, s) = g\}$ represents the set of states that can be reached from I with a cost of g. Typically, the search terminates whenever the next set of states to be expanded has a non-empty intersection with the set of goal states, also represented as a BDD $\mathcal{B}_G$, meaning that an optimal plan has been found. However, one can also choose to continue the search beyond that point to exhaust the set of reachable states.

On the other hand, symbolic backward search (regression) starts the search from the goal states, and applies the preimage operation until a non-empty intersection with the BDD that represents the initial state I is found. As a result, we obtain BDDs $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2$, where $\mathcal{B}_h = \{s \mid h^*(s, G) = h\}$ represents the set of states that can reach G with a cost of h. If the search is exhausted (ignoring I), this is equivalent to computing the perfect heuristic h^*_G.

The combination of these two searches forms a bidirectional search used by most modern symbolic search planners (Torralba et al. 2017; Speck, Mattmüller, and Nebel 2020). We will use $\text{SymbolicSearch}(I, \mathcal{O}, fw)$, and $\text{SymbolicSearch}(G, \mathcal{O}, bw)$ to denote the initialization of forward and backward searches, respectively. We will use STEP to denote a function that computes the image (preimage) of a forward (backward) search; FINISHED to denote a Boolean function that indicates if the given search has been exhausted, i.e., the image/preimage operation does not generate new non-expanded states; COMPLETE() to denote a function that runs the search until FINISHED is true; and COMPLETE($\mathcal{B}$) to denote a function that runs the search until reaching some $s \in \mathcal{B}$ or until the search is finished if no state in $\mathcal{B}$ is reachable.

Perimeter search (Dillenburg and Nelson 1994) consists of searching a perimeter around the goal. This idea has been used in the construction of heuristic functions for explicit-state search using a symbolic backward search, which is interrupted before termination (Kissmann and Edelkamp 2011; Torralba, Linares López, and Borrajo 2018). Given an unfinished backward search initialized with G where the sets of states $\mathcal{B}_0, \ldots, \mathcal{B}_p$ have been generated up to $h = p$, the set of closed states is $closed = \bigvee_{i \in [0,p]} \mathcal{B}_i$, i.e., the set of states for which the real goal distance is known. Then, the perimeter heuristic is $h^P_G(s) = i$ if $s \in \mathcal{B}_i$ and $h^P_G(s) = p + 1$ otherwise.h^P_G is a lower bound on the actual goal distance, i.e., $h^P_G(s) \leq h^*_G(s)$. As sets of states are generated with increasing values of h, we are certain that any remaining state will have a goal distance of at least $p + 1$. The function $search$.CLOSEDBDD() returns a BDD with the set of states closed by a given search. The function $search$.KNOWNBDD() returns a BDD with the set of states for which the exact distance is known (i.e., $\{s \mid h^P_G(s) = h^*_G(s)\}$). This is the same as $search$.CLOSEDBDD() if the search has not finished, and equal to the set of all states otherwise. The function $search$.CLOSEDADD() returns an ADD mapping each state to the corresponding perimeter heuristic, i.e., $\min_i \text{ADD}(\mathcal{B}_i, i, \infty)$.

Multi-goal Distance Functions

In order to compute planning centroid and minimum covering states, we consider the same setting as in (Pozanco et al. 2019; Karpas 2022) using the concept of planning tasks with multiple possible goal conditions (PMG).

Definition 1 (PMG) *A planning task with multiple possible goal conditions is defined as $\mathcal{P} = \langle V, I, \mathcal{O}, \mathcal{G} \rangle$, where V, I, and $\mathcal{O}$ are defined as in a SAS$^+$ planning task, and $\mathcal{G}$ is a set of* ***possible goal conditions****, where each possible goal condition $G \in \mathcal{G}$ is defined as in a SAS$^+$ planning task.*

Definition 2 (Planning centroids) *Given a PMG $\mathcal{P}$, the* ***centroids*** *are those reachable states $s \in \mathcal{S}_R$ that minimize the sum of optimal costs to the possible goals, i.e., $\sum_{G \in \mathcal{G}} h^*_G(s)$.*

Definition 3 (Planning minimum covering states) *Given a PMG $\mathcal{P}$, the* ***minimum covering states*** *are those reachable states $s \in \mathcal{S}_R$ that minimize the maximum optimal cost to any of the possible goals, i.e., $\max_{G \in \mathcal{G}} h^*_G(s)$.*

Whenever a goal $G \in \mathcal{G}$ is unreachable, then $h^*_G(s) = \infty$ for all reachable states, so $\sum_{G \in \mathcal{G}} h^*_G(s) =$

$\max_{G\in\mathcal{G}} h^*_G(s) = \infty$. In that case, any reachable state is a centroid and minimum covering state.

We introduce a general definition that covers these and other states that optimize a multi-goal distance function.

Definition 4 (Multi-goal distance function) *A **multi-goal distance function** ϕ is a function $(\mathbb{N}_0 \cup \{\infty\})^+ \mapsto \mathbb{N}_0 \cup \{\infty\}$ mapping each vector of goal-distances to a goal-distance.*

Definition 5 (ϕ-optimal states) *Given a PMG $\mathcal{P}$, and a multi-goal distance function ϕ, the set of **ϕ-optimal states** is the set of reachable states $s \in \mathcal{S}_R$ that minimize $\phi(h^*_{G_1}(s), \dots, h^*_{G_n}(s))$.*

This definition generalizes centroids (when ϕ is the sum), and minimum covering states (when ϕ is the maximum). An important property, that is satisfied by both centroid and minimum covering states is monotonicity, i.e., minimizing the distance to each of the goals is always desirable.

Definition 6 (Monotonic multi-goal distance functions) *A multi-goal distance function ϕ is **monotonic** if for any two distance vectors $v = \langle v_1, \dots, v_k \rangle$ and $v' = \langle v'_1, \dots, v'_k \rangle$ with $|v| = |v'|$, it holds that $(\forall_i v_i \leq v'_i) \implies \phi(v) \leq \phi(v')$.*

For example, the sum function of centroids is monotonic because, if the distance to the remaining goals is preserved, increasing the distance to one of the goals cannot decrease the overall sum of distances. However, this is not the case for other functions. For example, if we are interested on finding states with the same cost to all the goals, we could minimize $\phi(h^*_{\mathcal{G}}(s)) = \max_{G\in\mathcal{G}} h^*_G(s) - \min_{G\in\mathcal{G}} h^*_G(s)$. This is non-monotonic, as a goal distance of $\langle 10, 10 \rangle$ is preferable over $\langle 5, 0 \rangle$ even though the distance to both goals is larger.

Computing ϕ-Optimal States with Exhaustive Symbolic Search

The first algorithm we propose from our SMG family of algorithms is Symbolic Multi-Goal Exhaustive Search (SMGE), which is described in Algorithm 1. SMGE starts by performing a full symbolic forward search from the initial state and $|\mathcal{G}|$ symbolic backward searches, one from each possible goal $G \in \mathcal{G}$. All of these searches are independent, so they can be performed in any order. The forward search yields *fw*.CLOSEDBDD(), a BDD representing the set of reachable states $\mathcal{S}_R$. Each backward search yields BW_G.CLOSEDADD(), an ADD representing the h^*_G function that maps each state to its distance to the goal $G \in \mathcal{G}$.

Algorithm 1: Symbolic Multi-Goal Exhaustive Search (SMGE)

```
Input: PMG task $\mathcal{P} = \langle V, I, \mathcal{O}, \mathcal{G} \rangle$ and multi-goal
       distance function: $\phi$
Output: A $\phi$-optimal state $s_\phi$
1 fw ← SymbolicSearch($I, \mathcal{O}, fw$)
2 BW ← {SymbolicSearch($G, \mathcal{O}, bw$) | $G \in \mathcal{G}$}
3 for search ∈ {fw} ∪ BW do
4     search.COMPLETE()
5 $\mathcal{A}_\phi$ ← APPLY($\bigcup_{G\in\mathcal{G}}\{\text{BW}_G$.CLOSEDADD()$\}, \phi$)
6 $\mathcal{A}_\phi$ ← max($\mathcal{A}_\phi$, ADD(fw.CLOSEDBDD(), 0, ∞))
7 value ← $\min_s \mathcal{A}_\phi(s)$
8 if value = ∞ then return $I$
9 else return any $s_\phi \in \arg\min_s \mathcal{A}_\phi(s)$
```

The key step of the algorithm is in line 5 where all the ADDs are combined into a single ADD $\mathcal{A}_\phi$ that represents ϕ, i.e., it maps each state s to the $\phi(s)$ value. In the pseudocode, we slightly abuse notation and describe this step as using APPLY over a set of ADDs (recall that APPLY can be used only for a pair of ADDs). The implementation of this step depends on the function ϕ. The idea is to repeatedly use APPLY to pairs of ADDs in the set until obtaining the desired result. For example, let $X = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3\}$ be a set of ADDs. APPLY(X,+) would compute $(\mathcal{A}_1 + \mathcal{A}_2) + \mathcal{A}_3$. For some binary operations, such as the sum and the maximum, the order in which the ADDs are combined is irrelevant for the final result. However, this order might influence the size of the intermediate ADDs, and therefore the runtime of this step of the algorithm. Therefore, in our implementation, we use the same procedure used to aggregate BDDs within the symbolic search algorithm (see Algorithm 3 in (Torralba et al. 2017)).

Then, SMGE updates the $\mathcal{A}_\phi$ by setting to ∞ the value of unreachable states, i.e., those states that do not appear in the CLOSED list of the forward search. After that, it takes the minimum value of $\mathcal{A}_\phi$, and returns any state optimizing ϕ by obtaining a path from the root to the terminal node with that minimum value. A plan from I (to $\mathcal{G}$) can be reconstructed by analyzing the forward search's (backward searches') CLOSED lists.

SMGE can easily be applied to any other multi-goal distance function. Decomposing the computation of ϕ into pairwise operations in line 5 may be more complicated, but it is always possible, as the terminal nodes in intermediate ADDs can be labeled with any needed information. In this paper, we focus on centroids and minimum covering states, and leave the implementation of this step for other multi-goal distance functions as future work.

From Exhaustive to Perimeter Search

The main drawback of the exhaustive search is that it performs unnecessary computations by obtaining h^* to each goal from each state in the problem. Whenever ϕ is monotonic, some of these computations are not needed and we can stop the backward searches earlier.

As an example, consider the problem of computing the minimum covering state on a PMG $\mathcal{P}$ with 3 possible goals, where the solution to such a task is a single state s with $h^*_{G_1}(s) = 10$, $h^*_{G_2}(s) = 4$, and $h^*_{G_3}(s) = 3$. In this case, any state with h^* larger than 10 to any goal will not be part of the solution, so it is unnecessary to expand them as the previous exhaustive algorithm does. For example, it is irrelevant if $h^*_{G_2}(s') = 15$ or $h^*_{G_2}(s') = 20$, so we can stop all searches after exploring all states of distance 10 or less to the goal. However, this raises the following question: up to when do we need to keep exploring for any given ϕ function?

In this example, it is clear that we need to explore each search tree at least up to distances 10, 4, and 3, respectively.

Otherwise, we would not know the distance from the solution state to each goal. However, this is not sufficient to prove that the obtained solution is indeed optimal. At that point, we know there is no minimum covering state at distance 3 or below, but there could be a solution at distance 5 from each goal. Expanding all the backward searches up to distance 10 is not necessary. In this case, it could suffice to explore all states at distances 10, 9 and 3, if there is no reachable state at distance 9 or less in the first two searches. Alternatively, we could explore up to 10, 4 and 9, if there is no state at distance 9 or less in the first and third search.

To formalize when we have performed enough search to guarantee that we have found the desired ϕ-optimal state, we define the set of candidate states with respect to the perimeter heuristics of all backward searches and a set of potentially reachable states.

Definition 7 (Candidate States) *Let BW be a set of perimeter searches, and $h^P_{\mathcal{G}} = (h^P_{G_1}, \dots, h^P_{G_n})$ a vector of the perimeter heuristics to each goal $G \in \mathcal{G}$. Let $S \supseteq \mathcal{S}_R$ be an over-approximation of the set of reachable states. The set of **candidate states** of BW for some multi-goal distance function ϕ given S is the set of states $\arg\min_{s \in S} \phi(h^P_{\mathcal{G}}(s))$.*

This allows for early termination without completing all backward searches, whenever there is a candidate state that has been closed in all backward searches and is reachable according to the forward search.

Proposition 1 (Termination Criteria) *Let ϕ be a monotone multi-goal distance function. Let BW be a set of perimeter searches with a set of candidate states $\mathcal{B}_C$. Then, if there exists $s \in \mathcal{B}_C$ such that $s \in bw.\text{KNOWNBDD}()$ for all $bw \in BW$ and $s \in \mathcal{S}_R$, then s is ϕ-optimal.*

Proof: Since s is a candidate, then s has minimum $\phi(h^P_{\mathcal{G}})$. Since $s \in bw.\text{KNOWNBDD}()$ for all $bw \in$ BW, then $h^P_G(s) = h^*_G(s)$ for all goals $G \in \mathcal{G}$. Therefore $\phi(h^P_{\mathcal{G}}) = \phi(h^*_{\mathcal{G}})$. Finally, let $s' \in \mathcal{S}_R$ be any other state. As $h^P_G(s') \leq h^*_G(s')$, by monotonicity of ϕ, we have that $\phi(h^P_{\mathcal{G}}(s')) \leq \phi(h^*_{\mathcal{G}}(s'))$. So, we conclude that $\phi(h^*_{\mathcal{G}}) = \phi(h^P_{\mathcal{G}}) \leq \phi(h^P_{\mathcal{G}}(s')) \leq \phi(h^*_{\mathcal{G}}(s'))$. As this holds for all s', s is a ϕ-optimal state with respect to ϕ, i.e., $s \in S_\phi$. □

Algorithm 2 shows our second algorithm, Symbolic Multi-Goal Perimeter Search (SMGP). The algorithm exchanges information from the backward and forward searches, so it has a parameter, *dir*, that specifies which direction is explored at the beginning of the algorithm. In SMGP$_{fw}$, the forward search is performed at the beginning. In SMGP$_{bw}$, the forward search is delayed until there are some candidates which have been explored in all backward searches. The main advantage of SMGP$_{fw}$ is that one can prune all unreachable states from the backward searches, which may increase the efficiency. On the other hand, the forward search can terminate early in SMGP$_{bw}$, if one of the states in the first set of explored candidates is reachable.

At each step within the main loop (lines 7-23), the algorithm generates a set of candidates, $\mathcal{B}_C$. This is done by joining the perimeter heuristic ADDs from all the backward searches in a single ADD through the APPLY operation (line 7), as described in the previous section. Then, if the forward search has finished, all unreachable states are assigned a value of ∞. Next, $\mathcal{B}_C$ is assigned to the set of states minimizing such function (lines 10-13), i.e., the set of candidate states according to Definition 7, where S is the set of reachable states if the forward search has already been performed, or the set of all states otherwise.

After that, the algorithm checks the two termination criteria in Proposition 1. First, whether some candidates have been explored by all the backward searches (line 14). If that is not the case, the algorithm performs a step in one of the backward searches, so that the set of candidates changes in the next iteration. To that end, we select a search for which the value of some of the candidates is not yet known. Then, SMGP advances the easiest backward search in BW. We define it to be the one that is estimated to generate a lower number of nodes in the next step (lines 22 and 23).

When $\mathcal{B}_{expCand} \neq \bot$, some candidates satisfy the first termination condition. In that case, we check if they are reachable (lines 16-17). This only incurs in computation if the forward search has not finished yet, as otherwise unreachable states were already removed in line 9. If $\mathcal{B}_\phi$ is not empty, all states in that intersection are reachable and con-

Algorithm 2: Symbolic Multi-Goal Perimeter Search (SMGP$_{dir}$)

```
Parameter : $dir$ = forward (fw) or backward (bw)
Input: PMG task $\mathcal{P} = \langle V, I, \mathcal{O}, \mathcal{G} \rangle$ and monotone
       multi-goal distance function: $\phi$
Output: A $\phi$-optimal state $s_\phi$
1  fw ← SymbolicSearch($I, \mathcal{O}, fw$)
2  BW ← {SymbolicSearch($G, \mathcal{O}, bw$) | $G \in \mathcal{G}$}
3  if dir = "forward" then
4      fw.COMPLETE()
5      Prune $fw$.CLOSEDBDD() on all BW$_G$ ∈ BW
6  while true do
7      $\mathcal{A}_\phi$ ← APPLY($\bigcup_{G \in \mathcal{G}}$ {BW$_G$.CLOSEDADD()}, $\phi$)      ⎫
8      if fw.FINISHED() then                                                      ⎪
9          $\mathcal{A}_\phi$ ←                                                    ⎪
             max($\mathcal{A}_\phi$, ADD(fw.CLOSEDBDD(), 0, $\infty$))                  ⎬ Get candidates
10     value ← $\min_s \mathcal{A}_\phi(s)$                                              ⎪
11     if value = $\infty$ then                                                      ⎪
12         return $I$                                                            ⎪
13     $\mathcal{B}_C$ ← BDD($\mathcal{A}_\phi$, value)                                        ⎭
14     $\mathcal{B}_{expCand}$ ← $\mathcal{B}_C \wedge \bigwedge_{G \in \mathcal{G}}$ BW$_G$.KNOWNBDD()             ⎫
15     if $\mathcal{B}_{expCand} \neq \bot$ then                                           ⎪
16         fw.COMPLETE($\mathcal{B}_{expCand}$)                                          ⎬ Check termination
17         $\mathcal{B}_\phi$ ← $\mathcal{B}_{expCand}$ ∧ fw.CLOSEDBDD()                             ⎪
18         if $\mathcal{B}_\phi \neq \bot$ then                                          ⎪
19             return any $s_\phi \in \mathcal{B}_\phi$                                    ⎭
20     else                                                                   ⎫
21         BWC ← {BW$_G$ ∈ BW |                                                   ⎪
              ¬BW$_G$.KNOWNBDD() ≠ ⊥}                                             ⎬ Step bw
22         bw_to_advance ← PICKEASIEST(BWC)                                       ⎪
23         bw_to_advance.STEP()                                                   ⎭
```

tain a set of ϕ-optimal states. Therefore, SMGP terminates returning any of the states in that BDD (line 19). Otherwise, this process is repeated until a ϕ-optimal state is found.

Proposition 2 SMGP$_{dir}$ *always terminates and returns a ϕ-optimal state for any monotone function ϕ.*

Proof Sketch: Termination is guaranteed as at every iteration; either we compute the set of reachable states or some unfinished backward search will take a step. The number of steps of a backward search is bounded by the number of states in the planning task, so eventually all searches will finish. At that point, the goal distance to all states is known, so the algorithm will return a reachable state with minimum ϕ-value.

Whenever the algorithm terminates, it always returns a ϕ-optimal state. Note that all states in $\mathcal{B}_\phi$ are candidates (due to line 13), closed in all unfinished backward searches (due to line 14), and reachable (due to line 17). Therefore, all conditions of Proposition 1 are met. □

Just Using BDDs for Minimum Covering States

SMGP constructs an ADD mapping each state s to its corresponding $\phi(s)$ value, according to the perimeter heuristic. However, as we are only interested in the states with minimum ϕ-value, computing the exact value for states with larger ϕ-value is unnecessary. While avoiding incurring this computational overhead for any arbitrary ϕ is not straightforward, one can easily do this for minimum covering states. The main observation is that, if all backward searches have been performed up to a perimeter k, then there exists a minimum covering state at distance k or less of all goals iff the intersection of all closed lists is not empty.

The third algorithm we propose is Symbolic Multi-Goal Minimum Covering Search (SMGC), which leverages this observation to compute minimum covering states through only using BDDs (see Algorithm 3). As in SMGP, we have two variants that differ on whether the forward search is completed at the beginning of the algorithm, or delayed until a set of candidates is found. Similarly to SMGP, the algorithm first checks whether there is a candidate solution. But, in this case, this is done by intersecting the closed list of all the backward searches (line 7). If this intersection is not empty, $\mathcal{B}_{expCand}$ contains a set of candidate solutions, and the algorithm proceeds to check whether some of its states are reachable. If so, SMGC returns any state in that BDD (line 12). Otherwise, when $\mathcal{B}_{expCand}$ is empty, the algorithm (lines 13-18) progresses all the searches that have not yet finished and whose next step has a lowest g-value. This simplifies the algorithm, as it ensures that all the searches have always explored up to the same perimeter.

Algorithm 3: Symbolic Multi-Goal Minimum Covering Search (SMGC$_{dir}$)

```
Parameter : $dir$ = forward (fw) or backward (bw)
Input: PMG task $\mathcal{P} = \langle V, I, \mathcal{O}, \mathcal{G} \rangle$
Output: A minimum-covering state $s_\phi$
1  fw ← SymbolicSearch($I, \mathcal{O}, fw$)
2  BW ← {SymbolicSearch($G, \mathcal{O}, bw$) | $G \in \mathcal{G}$}
3  if dir = "forward" then
4      fw.COMPLETE()
5      Prune $fw$.CLOSEDBDD() on all BW$_G \in$ BW
6  while true do
7      $\mathcal{B}_{expCand} \leftarrow \bigwedge_{G \in \mathcal{G}}$ BW$_G$.CLOSEDBDD()
8      if $\mathcal{B}_{expCand} \neq \bot$ then
9          fw.COMPLETE($\mathcal{B}_{expCand}$)
10         $\mathcal{B}_\phi \leftarrow \mathcal{B}_{expCand} \wedge$ fw.CLOSEDBDD()
11         if $\mathcal{B}_\phi \neq \bot$ then
12             return any $s_\phi \in \mathcal{B}_\phi$
13     $k \leftarrow \min_{\text{BW}_G \in \text{BW}}$ BW$_G$.NEXTLAYER()
14     BWC ← {BW$_G \in$ BW |
           ¬BW$_G$.FINISHED() ∧ BW$_G$.NEXTLAYER() $= k$}
15     if BWC = ∅ then
16         return $I$
17     foreach BW$_G \in$ BWC do
18         BW$_G$.STEP()
```

Experiments

Approaches. We implemented our SMG algorithms on top of Symbolic Fast Downward (Torralba et al. 2017)[1], and compare them against the two other approaches in the literature to compute centroids and minimum covering states: GRS and COMP. GRS (Pozanco et al. 2019)[2] uses the Fast Downward planner (Helmert 2006) with the A* search algorithm and the LM-cut heuristic (Helmert and Domshlak 2009) to perform exhaustive search in the original planning task with multiple possible goals $\mathcal{P}$. COMP (Karpas 2022)[3] compiles $\mathcal{P}$ into a standard planning task Π. We use two planners to solve Π: A* with the LM-cut heuristic as proposed by Karpas (2022), COMP$_{lm}$; and the symbolic bi-directional configuration of Symbolic Fast Downward, COMP$_{sb}$, so we make sure any performance improvement of the SMG algorithms does not come from using a different planner.

Benchmarks and Reproducibility. We use the benchmark set used by Pozanco et al. (2019) and Karpas (2022)[4], which consists of four planning domains (BLOCKS, FERRY, GRIPPER, and LOGISTICS), and one grid path-finding domain. The planning domains were adapted from standard IPC benchmarks, and the grid path-finding domains consists of 20×20 grids with a different percentage of obstacles ($5\%, 10\%, 15\%$, and 20%). Each domain has 10 problem instances, for a total of 80 problems equally split between IPC and grid domains. We refer to this benchmark as SMALL, as most of the tasks are small planning tasks with few possible goals. For example, all BLOCKS instances contain only 5 blocks and 3 goals. To better analyze the scalability of the algorithms, we introduce 80 tasks of increasing difficulty in each domain. In BLOCKS, we generated these instances by

[1]https://gitlab.com/atorralba/fast-downward-symbolic

[2]https://github.com/apozanco/GRS_0.1

[3]https://github.com/karpase/grscompilation

[4]We exclude HANOI, which only contained tasks with one goal.

creating random configurations of 6, 8, 10 or 12 blocks, and having 2, 4, 8 or 16 possible goals (words) to be formed by stacking the blocks. In GRID, we generated 10×10, 20×20, 40×40 and 80×80 grids, with 2, 4, 8 or 16 possible goals (agent's locations). For FERRY, GRIPPER and LOGISTICS, we selected the 20 first problems available at the Planning Domains repository[5], and created 4 different instances for each problem by generating between 2 and 5 possible goals. These goals are sets of cars/balls/packages being delivered at different locations/rooms/cities.

Experiments were run on an Intel Xeon E5-2666 v3 CPU @ 2.90GHz x 8 processors with a 8GB memory bound and a time limit of 1800s. Code, benchmarks, and results are publicly available (Pozanco, Torralba, and Borrajo 2024).

Coverage Analysis. Table 1 shows the coverage of each approach, i.e., the number of problems they solve. Regarding centroids, COMP$_{lm}$ solves more than twice the tasks solved by GRS in the small instances, as previously reported by Karpas (2022). This performance difference is emphasized in the larger tasks, where GRS solves only 22 tasks compared to the 257 for which COMP$_{lm}$ can compute a centroid. Solving the compiled tasks using a symbolic planner (COMP$_{sb}$) offers a worse performance across all domains, highlighting the fact that the reformulated task does not suit symbolic planners particularly well. Our baseline approach that performs exhaustive search (SMGE) is already very competitive with the previous state-of-the-art COMP$_{lm}$, achieving significantly higher coverage in BLOCKS, FERRY, and LOGISTICS. However, SMGE is worse in GRIPPER and fails to solve almost any instance in the GRID domain, where it can only solve 1 out of 40 problems. This is because this arguably ill-defined domain uses a `(free ?c)` predicate to denote that a given cell is free and the agent can move to it, even though in all instances there is a single agent so the precondition is always redundant. Having this predicate turns GRID tasks into a challenging domain for approaches using backward search, since the set of reachable states is orders of magnitude smaller than the set of states. For example, assuming a 20×20 grid without obstacles, the problem will have only 400 reachable states, but more than 2^{400} total states that backward searches could potentially consider. This is also the reason why SMGP$_{bw}$ obtains similar results, although it outperforms the exhaustive algorithm in larger tasks, being able to solve 45 tasks more. In fact, if we leave GRID aside, SMGP$_{bw}$ would be the best performing algorithm, solving 1 task more than SMGP$_{fw}$, the winner across the benchmark with a coverage of 415. By first computing the reachable states and using that information to prune the backward searches, SMGP$_{fw}$ is able to solve most of the GRID tasks, as well as showing a good performance in the other domains.

Similar conclusions can be drawn from the minimum covering states results (right side of Table 1). In this case, SMG approaches outperform the compilation approach by even larger margins, with the best performing SMGP variant solving 421 tasks compared to the 187 solved by COMP$_{lm}$. The compilation to compute minimum covering states is more involved than the centroids one, since it needs to discretize numerical variables. On the other hand, SMG algorithms achieve slightly better results when computing minimum covering states, with only 17% of problems where the opposite is true. The reason is that combining the ADDs of the backward searches by taking their maximum generates a smaller joint ADD than when these ADDs are combined by taking their sum, as the number of possible terminal node values is larger in the latter case. The performance of the SMGC algorithms is very similar to their SMGP counterparts.

Runtime Analysis. First, we compare the execution time of some of the best variants of COMP and SMG. These results are shown in Figure 3a, where we compare SMGP$_{fw}$ (x axis) and COMP$_{lm}$ (y axis) when computing minimum covering states. In this case, the majority of the tasks were solved faster by SMGP$_{fw}$ than by the compilation, i.e., points fall above the diagonal line in the plot. We can observe some trends across domains. For example, SMGP$_{fw}$ is consistently faster than COMP$_{lm}$ in BLOCKS and GRID instances, while the results in the other domains depend on the task at hand. There is a cluster of simple problems that COMP$_{lm}$ can solve in less than a second, while SMGP$_{fw}$ requires up to 1 second to solve them. Our algorithms initialize some data structures before starting the search, which slightly delays SMGP algorithms in some simple tasks. The results for centroids are similar, with SMGP$_{fw}$ being significantly faster.

Forward SMG variants obtained higher coverage scores mainly due to GRID, but the backward variants were able to solve more problems in domains such as LOGISTICS and BLOCKS. Figure 3b compares the execution time of SMGP$_{bw}$ and SMGP$_{fw}$ when computing centroids. As we can see, SMGP$_{bw}$ is faster than SMGP$_{fw}$ in most cases except for GRID. This was expected, as SMGP$_{bw}$ does not need to compute all the set of reachable states as SMGP$_{fw}$ does, which might be demanding in some problems. In particular, starting the search backwards until a candidate centroid is found is faster in 70% of the tasks. This percentage raises up to a $\approx$ 90% if we do not consider GRID, re-emphasizing SMGP$_{bw}$ as the go-to algorithm for many IPC domains such as BLOCKS where (i) most of the states in the planning task are reachable; and (ii) the number of reachable states is large.

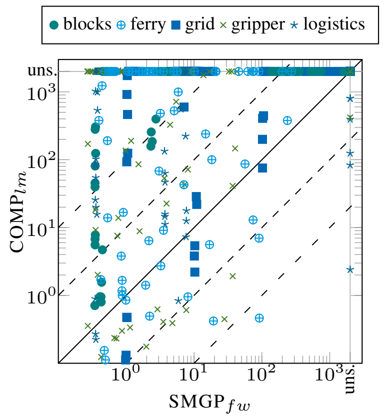
The coverage difference between the SMGP variants, which use ADDs and BDDs and progress only one backward search at a time, and the SMGC variants, which only use BDDs and progress multiple backward searches simultaneously, was negligible. Figure 3c compares the execution time of their backward versions when computing minimum covering states. As we can see, both approaches have very similar execution times. The main differences appear in BLOCKS tasks, where SMGC$_{bw}$ is slower in most of the tasks. These similar runtimes are explained by two counteracting factors. On the one hand, SMGC variants should be faster, as they do not need to build and reason over ADDs. On the other hand, SMGP variants should be faster, as they typically require to advance a lower number of backward searches.

Scalability Analysis. We also analyzed the performance of the algorithms in detail in the BLOCKS domain, where

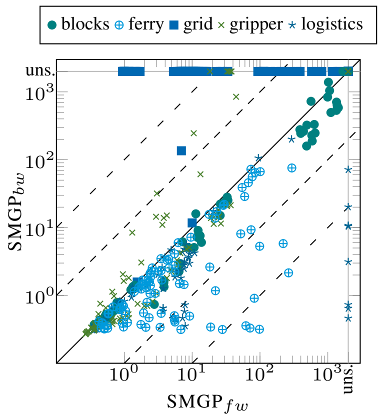
[5]https://github.com/AI-Planning/classical-domains

	Centroid						Minimum Covering							
Domain	GRS	COMP$_{lm}$	COMP$_{sb}$	SMGE	SMGP$_{bw}$	SMGP$_{fw}$	GRS	COMP$_{lm}$	COMP$_{sb}$	SMGE	SMGP$_{bw}$	SMGP$_{fw}$	SMGC$_{bw}$	SMGC$_{fw}$
BLOCKS (10)	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**
FERRY (10)	0	9	2	**10**	**10**	**10**	0	7	6	**10**	**10**	**10**	**10**	**10**
GRIPPER (10)	1	**10**	**10**	**10**	**10**	**10**	2	**10**	8	**10**	**10**	**10**	**10**	**10**
LOGISTICS (10)	5	**10**	4	**10**	**10**	**10**	7	**10**	**10**	**10**	**10**	**10**	**10**	**10**
GRID (40)	18	39	1	1	2	**40**	18	1	1	1	2	**40**	2	**40**
SMALL (80)	34	78	27	41	42	**80**	37	38	35	41	42	**80**	42	**80**
BLOCKS (80)	2	30	10	45	59	**60**	2	9	10	45	**68**	60	62	60
FERRY (80)	0	46	27	**80**	**80**	**80**	0	39	31	**80**	**80**	**80**	**80**	**80**
GRIPPER (80)	2	74	32	46	70	**75**	2	55	25	46	71	75	72	**77**
LOGISTICS (80)	3	57	32	64	**70**	62	3	27	26	64	**72**	64	**72**	64
GRID (80)	15	50	16	0	1	**58**	17	19	15	0	5	**60**	6	**60**
LARGE (400)	22	257	117	235	280	**335**	24	149	107	235	296	339	292	**341**
TOTAL (480)	56	335	144	276	322	**415**	61	187	142	276	338	419	334	**421**

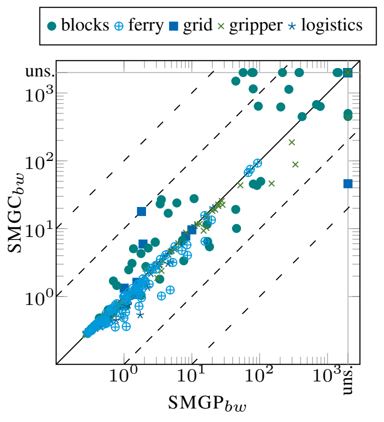
Table 1: Approaches' coverage in computing centroids and minimum covering states. Bold figures indicate best performance.



(a) Minimum covering states: SMGP$_{fw}$ vs COMP$_{lm}$.



(b) Centroids: SMGP$_{fw}$ vs SMGP$_{bw}$.



(c) Minimum covering states: SMGP$_{bw}$ vs SMGC$_{bw}$.

Figure 3: Pairwise comparisons of execution time. Each point in the plot corresponds to a problem of our joint benchmark (480 tasks). Points above the diagonal indicate that the approach in the x axis is faster than the approach in the y axis.

we generated tasks with increasing complexity by varying the number of blocks (size of the problem) and the number of words to build (number of possible goals). Our SMG approaches scale much better than the compilation ones in both criteria, and they are specially robust with respect to the number of goals.

Conclusions and Future Work

In this paper we introduced Symbolic Multi-Goal (SMG), a family of symbolic bidirectional search algorithms to compute centroids and minimum covering states. Experimental results in existing and novel benchmarks show that our algorithms outperform current approaches both in coverage and execution time. These results establish our algorithms as a new state-of-the-art technique for this task. The decision of which SMG algorithm should be used greatly depends on the task at hand. Forward alternatives are the best option when the number of reachable states is small compared to all states in the planning task, as it can prune many states and simplify the backward searches. Otherwise, backward alternatives are the best choice.

There are also many interesting research avenues. On the one hand, there are several ways to further improve SMGP's performance. First, SMGP advances the backward search whose next step is estimated to be easier. In future work, we would like to explore other less greedy criteria to make this decision. Second, we could over-approximate the set of reachable states, e.g., using mutexes (Alcázar and Torralba 2015; Fiser and Komenda 2018), to increase performance in the variants that prioritize backward search, which tend to be faster. Third, we would like to test whether some candidate state is reachable using heuristics, without conducting an exhaustive forward search. However, this requires computing heuristics wrt. a set of goal states (our candidate set) represented as a BDD. Currently, this is only possible for admissible heuristics derived with symbolic search (Torralba et al. 2016; Torralba, Linares López, and Borrajo 2018), which are not competitive for satisficing planning. Finally, we focused on two ϕ functions that had already been defined and proved useful in the literature. In future work we will consider other ϕ-optimal states, e.g. those that are at the same distance from all goals, or that are as far as possible from the goals.

Acknowledgements

This paper was prepared for informational purposes in part by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("J.P. Morgan") and is not a product of the Research Department of J.P. Morgan. J.P. Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In Brafman, R.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 2–6. AAAI Press.

Bäckström, C.; and 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.; and Somenzi, F. 1997. Algebraic Decision Diagrams and Their Applications. *Formal Methods in System Design*, 10(2–3): 171–206.

Bryant, R. E. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8): 677–691.

Burns, E.; Benton, J.; Ruml, W.; Yoon, S.; and Do, M. 2012. Anticipatory On-line Planning. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 333–337. AAAI Press.

Dillenburg, J. F.; and Nelson, P. C. 1994. Perimeter Search. *Artificial Intelligence*, 65: 165–178.

Fiser, D.; and Komenda, A. 2018. Fact-Alternating Mutex Groups for Classical Planning. *J. Artif. Intell. Res.*, 61: 475–521.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Karpas, E. 2022. A Compilation Based Approach to Finding Centroids and Minimum Covering States in Planning. In Thiébaux, S.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, 174–178. AAAI Press.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal Recognition Design. In Chien, S.; Fern, A.; Ruml, W.; and Do, M., eds., *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 154–162. AAAI Press.

Kissmann, P.; and Edelkamp, S. 2011. Improving Cost-Optimal Domain-Independent Symbolic Planning. In Burgard, W.; and Roth, D., eds., *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, 992–997. AAAI Press.

Pozanco, A.; E-Martín, Y.; Fernández, S.; and Borrajo, D. 2019. Finding Centroids and Minimum Covering States in Planning. In Lipovetzky, N.; Onaindia, E.; and Smith, D. E., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, 348–352. AAAI Press.

Pozanco, A.; Torralba, Á.; and Borrajo, D. 2024. Code, Benchmarks and Experiment Data for the ICAPS 2024 Paper "Computing Planning Centroids and Minimum Covering States using Symbolic Bidirectional Search". https://doi.org/10.5281/zenodo.10834945.

Price, A.; Pereira, R. F.; Masters, P.; and Vered, M. 2023. Domain-Independent Deceptive Planning. In Agmon, N.; An, B.; Ricci, A.; and Yeoh, W., eds., *Proceedings of the 2023 International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2023)*, 95–103. ACM.

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

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In Conitzer, V.; and Sha, F., eds., *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 9967–9974. AAAI Press.

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

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *Artificial Intelligence*, 242: 52–79.

Torralba, Á.; Gnad, D.; Dubbert, P.; and Hoffmann, J. 2016. On State-Dominance Criteria in Fork-Decoupled Search. In Kambhampati, S., ed., *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3265–3271. AAAI Press.

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