Symbolic Greedy Best-First Search with Operator-Potential Heuristics

Daniel Fišer, Álvaro Torralba

Aalborg University, Denmark
danfis@danfis.cz, alto@cs.aau.dk

Abstract

Symbolic search has long been recognized as one of the state-of-the-art techniques for exhaustive state-space exploration and cost-optimal planning, leveraging compact state representations and efficient set-based operations. Despite its success in optimal planning, symbolic search has not traditionally been considered suitable for satisficing search, where heuristic guidance is paramount. In this paper, we investigate how to perform symbolic satisficing search aimed at efficiently finding good-quality solutions without guaranteeing optimality. We show that operator-potential heuristics are effective at guiding the search. Empirical results demonstrate that symbolic greedy best-first search with operator-potential heuristics is competitive with explicit state-space search.

1 Introduction

Symbolic search is a technique for state-space exploration where sets of states are compactly represented and manipulated using Binary Decision Diagrams (BDDs) (Bryant 1986). Originally devised in model-checking (McMillan 1993), symbolic search has been applied with huge success for cost-optimal classical planning (Edelkamp and Helmert 2001; Edelkamp and Kissmann 2009; Edelkamp, Kissmann, and Torralba 2015), where it is the core of several state-of-the-art planners in the last decade (Torralba et al. 2014, 2016; Fišer, Torralba, and Hoffmann 2024; Speck, Seipp, and Torralba 2025). Furthermore, the success of symbolic search extends beyond optimal planning, with successful results on many other planning-related problems that require an exhaustive exploration of the state space, such as proving unsolvability (Torralba 2016), over-subscription planning (Speck and Katz 2021), and many others (Cimatti et al. 2003; Kissmann and Edelkamp 2010; Speck, Mattmüller, and Nebel 2020; Torralba et al. 2021; Behnke and Speck 2021; Pozanco, Torralba, and Borrajo 2024).

However, to the best of our knowledge, symbolic search has yet to be applied to one of the most prominent forms of planning: satisficing planning, i.e. finding a plan as good as possible but not necessarily optimal. While on the surface satisficing and optimal planning look very similar, the state of the art heuristic search algorithms for solving them are typically very different. Since proving optimality is no longer necessary, it suffices to expand the states along a path from the initial state to the goal. Consequently, state-of-the-art satisficing planners rely on Greedy Best-First Search (GBFS) with multiple heuristics (Richter and Westphal 2010) to steer exploration towards the goal.

At first glance, this suggests that symbolic search may not be suitable for satisficing planning. Its primary strength lies in efficient exhaustive exploration and this is no longer required in greedy search. Also, despite decades of research on how to utilize heuristics in symbolic search (Hansen, Zhou, and Feng 2002; Jensen, Veloso, and Bryant 2008; Kissmann and Edelkamp 2011), their combination remains non-trivial and faces multiple challenges. For example, symbolic search is strictly superior to explicit-state search when no heuristics are used (Edelkamp and Kissmann 2008; Speck and Helmert 2025), but this is no longer the case with heuristics and even perfect heuristics can be detrimental for the performance of symbolic search (Speck, Geißer, and Mattmüller 2020). Furthermore, some widely-used heuristics for satisficing planning such as delete-relaxation (Bonet and Geffner 2001; Hoffmann and Nebel 2001) are inherently per-state computations, making them incompatible with symbolic search algorithms, which must evaluate large sets of states simultaneously. Recent work by Fišer, Torralba, and Hoffmann (2024) demonstrated the effectiveness of operator-potential heuristics in symbolic search for optimal planning; however, their applicability to satisficing planning remains unexplored.

In this paper, we investigate the possibilities of symbolic heuristic search for satisficing planning. To this end, we introduce a bi-directional symbolic GBFS with operator-potential heuristics, and propose several enhancements such as a simplified termination condition (since optimality guarantees are no longer required) and a lazy image computation. We show that current (operator-)potential heuristics (Pommerening et al. 2015; Fišer, Torralba, and Hoffmann 2024), despite being admissible heuristics, are competitive with other popular inadmissible heuristics for satisficing planning such as FF (Hoffmann and Nebel 2001). We also show that the goal-counting heuristic is in fact a potential heuristic, and can thus be used in symbolic search. Overall, our experiments show that symbolic GBFS with operator-potential and goal-counting heuristics is a competitive satisficing planner, on par with other vanilla explicit-state heuristic search planners, and complementary to the state of the art.

2 Background

An **FDR planning task** Π is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$. $\mathcal{V}$ is a finite set of **variables**, each $V \in \mathcal{V}$ has a finite **domain** $\mathrm{dom}(V)$. A **fact** $\langle V, v \rangle$ is a pair of a variable V and one of its values $v \in \mathrm{dom}(V)$. $\mathcal{F} = \{\langle V, v \rangle \mid V \in \mathcal{V}, v \in \mathrm{dom}(V)\}$ denotes the set of all facts. A **partial state** p is a variable assignment over some variables $\mathcal{V}(p) \subseteq \mathcal{V}$. We write $p[V]$ for the value assigned to the variable $V \in \mathcal{V}(p)$ in p. We identify p with the set of facts it contains, i.e., $p = \{\langle V, p[V] \rangle \mid V \in \mathcal{V}(p)\}$. A partial state s is a **state** if $\mathcal{V}(s) = \mathcal{V}$. I is an **initial state**. G is a partial state called **goal**, and a state s is a **goal state** if $G \subseteq s$. $\mathcal{S}$ denotes the set of all states.

$\mathcal{O}$ is a finite set of **operators**, each $o \in \mathcal{O}$ has a **precondition** pre_o, **prevail condition** prv_o, and **effect** eff_o, which are partial states, and a cost $\mathrm{cost}(o) \in \mathbb{R}_0^+$. For every $o \in \mathcal{O}$ it holds that $\mathcal{V}(\mathrm{pre}_o) \subseteq \mathcal{V}(\mathrm{eff}_o)$ and $\mathcal{V}(\mathrm{prv}_o) \cap \mathcal{V}(\mathrm{eff}_o) = \emptyset$, i.e., preconditions are defined only over affected variables and prevail conditions only over unaffected variables. We also assume that $\mathrm{pre}_o[V] \neq \mathrm{eff}_o[V]$ for every $V \in \mathcal{V}(\mathrm{pre}_o)$. An operator o is **applicable** in a state s iff $\mathrm{prv}_o \cup \mathrm{pre}_o \subseteq s$. The **resulting state** of applying an applicable operator o in a state s is another state $o[\![s]\!]$ s.t. $o[\![s]\!][V] = \mathrm{eff}_o[V]$ for every $V \in \mathcal{V}(\mathrm{eff}_o)$, and $o[\![s]\!][V] = s[V]$ for every $V \in \mathcal{V} \setminus \mathcal{V}(\mathrm{eff}_o)$.

A planning task Π is in Transition Normal Form (TNF) if (i) $\mathcal{V}(\mathrm{pre}_o) = \mathcal{V}(\mathrm{eff}_o)$ for every $o \in \mathcal{O}$ and (ii) the goal is a state. Here, we are interested only in the first condition, so we say that Π is **normalized** if $\mathcal{V}(\mathrm{pre}_o) = \mathcal{V}(\mathrm{eff}_o)$ for all operators $o \in \mathcal{O}$.

Every planning task can be normalized so that it grows only polynomially (Pommerening and Helmert 2015; Fišer, Horčík, and Komenda 2020). Unfortunately, this transformation turns out to be detrimental to symbolic search as pointed out by Fišer, Torralba, and Hoffmann (2024). However, they proposed a straightforward "multiplication" method having a good synergy with symbolic search methods: for every $o \in \mathcal{O}$, every $V \in \mathcal{V}(\mathrm{eff}_o) \setminus \mathcal{V}(\mathrm{pre}_o)$, and every $v \in \mathrm{dom}(V)$, create the corresponding operator o' with $\langle V, v \rangle \in \mathrm{pre}_{o'}$ and repeat this process until $\mathcal{V}(\mathrm{pre}_o) = \mathcal{V}(\mathrm{eff}_o)$ for all operators o.

Given a non-negative integer n, $[n]$ denotes $\{1, \ldots, n\}$ and $[0]=\emptyset$. A sum over an empty set is considered to be zero.

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 $s_i = o_i[\![s_{i-1}]\!]$ for $i \in [n]$. The resulting state of this application is $\pi[\![s_0]\!] = s_n$ and $\mathrm{cost}(\pi) = \sum_{i \in [n]} \mathrm{cost}(o_i)$ is the cost of π. We also allow empty sequences π applicable in all states s and $\pi[\![s]\!] = s$.

A sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ is called an **s-t-path** if there exist states s and t such that π is applicable in s and $\pi[\![s]\!] = t$; π is called an **s-plan** if it is applicable in the state s and $\pi[\![s]\!]$ is a goal state; and an I-plan is called a **plan**. An s-t-path (s-plan, plan) π is called **optimal** if its cost is minimal among all s-t-paths (s-plans, plans).

A state s is **forward reachable** if there exists an I-s-path, and s is **backward reachable** if there exists an s-plan. A **forward heuristic** (**backward heuristic**) estimates the cost of optimal s-plans (I-s-paths) for states s. The **optimal** forward (backward) heuristic returns the cost of optimal s-plans (I-s-paths) and ∞ if there are no such s-plans (I-s-paths). We allow heuristics h to return negative numbers (heuristic values, h-values) as is usual in works on potential heuristics; negative h-values can be treated as zeros during the search. A forward (backward) heuristic is called **admissible** if it underestimates the optimal forward (backward) heuristic. When clear from the context or the distinction is unimportant, we omit forward and backward qualifiers when speaking about heuristics.

We call a heuristic **state-dependent** if it is a function from states to numbers, and **path-dependent** if it is a mapping from sequences of operators to numbers, i.e., a mapping from I-s-paths in forward heuristics, and from s-plans in backward heuristics. Clearly, if a path-dependent forward (backward) heuristic returns the same value for every I-s-path (s-plan) it can be understood as state-dependent as well.

2.1 Operator-Potential Heuristics

Potential heuristics (Pommerening et al. 2015; Pommerening, Helmert, and Bonet 2017) are defined as weighted sums over a set of simple state features that correspond to a conjunction of facts. The dimension of a feature is the number of facts in the corresponding conjunction. We consider here the simplest variant, one-dimensional potential heuristics (also called atomic), where all features are single facts: A **potential function** $\mathtt{P} : \mathcal{F} \mapsto \mathbb{R}$ maps each fact to a numerical value, and a **potential heuristic** $h^{\mathtt{P}}$ for $\mathtt{P}$ maps each state $s \in \mathcal{S}$ to the sum of potentials of facts in s, i.e., $h^{\mathtt{P}}(s) = \sum_{f \in s} \mathtt{P}(f)$. They are state-dependent and designed to be used in the forward search direction.

Potential functions $\mathtt{P}$ are computed as solutions to Linear Programs (LPs) with constraints expressing consistency ($h(s) \leq h(s') + c(o)$ for all forward reachable states s, operators o applicable in s, and $o[\![s]\!] = s'$), and goal-awareness ($h(s) \leq 0$ for all forward reachable goal states s) of the induced potential heuristics $h^{\mathtt{P}}$. Objective functions of these LPs determine different types of potential heuristics and there are several different types described in prior works (Pommerening et al. 2015; Pommerening, Helmert, and Bonet 2017; Fišer, Horčík, and Komenda 2020).

Operator-potential heuristics (Fišer, Torralba, and Hoffmann 2022a, 2024) turn (atomic) potential heuristics into path-dependent heuristics by associating each operator with the change induced by the potential heuristic when applying the operator: given a potential heuristic $h^{\mathtt{P}}$, we can construct an **operator-potential function** $\mathtt{Q} \colon \mathcal{O} \mapsto \mathbb{R}$ so that the operator-potential forward heuristic $h^{\mathtt{Q}}_{\mathrm{fw}}$ is computed as $h^{\mathtt{Q}}_{\mathrm{fw}}(\pi) = h^{\mathtt{P}}(I) + \sum_{i \in [n]} \mathtt{Q}(o_i)$ for every I-s-path $\pi = \langle o_1, \ldots, o_n \rangle$, and $h^{\mathtt{Q}}_{\mathrm{fw}}(\pi) \leq h^{\mathtt{P}}(s)$, i.e., $h^{\mathtt{Q}}_{\mathrm{fw}}$ is admissible when $h^{\mathtt{P}}$ is admissible. We can make a similar construction for the operator-potential backward heuristic $h^{\mathtt{Q}}_{\mathrm{bw}}$ so that $h^{\mathtt{Q}}_{\mathrm{bw}}(\pi) = h^{\mathtt{P}}(I) + \sum_{i \in [n]} \mathtt{Q}(o_i)$ for every s-plan $\pi = \langle o_1, \ldots, o_n \rangle$, and h_{bw} is admissible when $h^{\mathtt{P}}$ is admissible. Moreover, Fišer et al. (2022a; 2022b; 2024) showed that for normalized tasks, $h^{\mathtt{Q}}_{\mathrm{fw}}(\pi)$ is state-dependent and equal to $h^{\mathtt{P}}(s)$ for every I-s-path π, and $h^{\mathtt{Q}}_{\mathrm{bw}}(\pi)$ can be made state-dependent for every s-plan π by subtracting $h^{\mathtt{P}}(\pi[\![s]\!])$.

2.2 Symbolic Search

While explicit state-space search algorithms operate on individual states, symbolic search (McMillan 1993) works on sets of states compactly represented as Binary Decision Diagrams (BDDs) (Bryant 1986). BDDs are directed acyclic graph data-structures to represent Boolean functions $\{0,1\}^n \mapsto \{0,1\}$. The size of a BDD refers to the number of nodes it contains. A set of states $S \subseteq \mathcal{S}$ is represented as a BDD via its characteristic function $\mathcal{S} \mapsto \{0,1\}$ assigning 1 to states that belong to S and 0 to the rest. This assumes a binary encoding of states. We use the standard representation and variable ordering used in previous works (Kissmann and Edelkamp 2011; Torralba et al. 2017). The advantage is that BDDs can be exponentially smaller than the number of states in the set they represent. And we can leverage operations on BDDs to operate with sets of states without enumerating them one by one. For example, the union ($\cup$), intersection ($\cap$), and complement of sets of states can efficiently be performed as the disjunction ($\vee$), conjunction ($\wedge$), and negation ($\neg$) of their characteristic functions, respectively.

To perform symbolic search, the operators of the planning task are represented as **transition relations** (TRs), also using BDDs. A TR of an operator o is a characteristic function $T_o : \mathcal{S} \times \mathcal{S} \mapsto \{0,1\}$ that represents all pairs of states $\langle s, o[\![s]\!]\rangle$ such that o is applicable in s. Having a TR T_o for every operator $o \in \mathcal{O}$, we can construct a TR representing all operators with the same cost c as $T_c = \bigvee_{o\in\mathcal{O},\mathrm{cost}(o)=c} T_o$. That is, T_c represents all pairs of states $\langle s, s'\rangle$ such that s' can be reached from s by applying some operator with cost c. As the size of T_c may be exponential in the number of operators with cost c, in practice, it is often a good idea to use **disjunctive partitioning** to keep the size at bay (Jensen, Veloso, and Bryant 2008; Torralba, Edelkamp, and Kissmann 2013; Torralba et al. 2017). Moreover, mutexes can be used for a more accurate approximation of reachable states (Torralba and Alcázar 2013; Torralba et al. 2017).

Given a BDD S representing a set of states and a TR T_c, $\mathrm{image}(S, T_c)$ computes the set of successor states reachable from any state in S by applying any operator represented by T_c. To perform forward search, we iteratively apply the **image** operation starting with the BDD representing $\{I\}$. By using a separate TR per operator cost c, one can easily keep track of the cost of reaching a state. If S_g represents a set of states reachable with cost g, then all states in $\mathrm{image}(S_g, T_c)$ are reachable with a cost of $g + c$. By repeatedly applying this operation, one can enumerate all states, classified into sets $S_0, S_1, \ldots$ according to their distance from the initial state. Whenever the representation of each S_g is compact, one can get exponential gains with respect to explicit-state search (Edelkamp and Kissmann 2008). In the backward direction, one can start with the BDD representing all goal states and use the **pre-image** operation, i.e., $\text{pre-image}(S, T_c)$ computes the set of all predecessor states from which a state in S can be reached with a transition in T_c. Torralba et al. (2017) provide a comprehensive description of how to efficiently implement image operations.

Several symbolic search algorithms differ on how they classify the sets of states generated during the search, and in what order they are expanded. Symbolic uniform-cost search classifies all states in the open list by their g-value. With heuristics, several variants of A* (Hart, Nilsson, and Raphael 1968) have been used for optimal planning. BDDA* (Edelkamp and Reffel 1998) and GHSETA* (Jensen, Veloso, and Bryant 2008) organize the states in the open list into sets with the same g and h value, represented as a BDD $S_{g,h}$, whereas FSETA* classifies them into sets of states, S_{g+h}, with the same f-value.

At each iteration one of the sets is selected for expansion, computing the image with respect to all transition relations and inserting the generated set of states in the open list. This requires to classify the resulting sets of states according to their heuristic value. We follow the approach proposed by GHSETA*, where we have a TR $T_{c,q}$ representing transitions from s to s' with cost c where $q = h(s') - h(s)$ is the difference between heuristic values. With this approach, computing g and h-values of successor states is straightforward: $\mathrm{image}(S_{g,h}, T_{c,q})$ directly results in the BDD $S_{g+c,h+q}$ representing all successor states of $S_{g,h}$ with g-value $g + c$ and h-value $h + q$. A caveat is that, for most heuristics, splitting the transition relation according to the difference in heuristic value is intractable. But, for operator-potential heuristics, we can simply compute $T_{c,q} = \bigvee_{o\in\mathcal{O},\mathrm{cost}(o)=c,\mathtt{Q}(o)=q} T_o$.

3 Symbolic Greedy Best-First Search

Algorithm 1 describes bi-directional symbolic GBFS with operator-potential heuristics. The algorithm is essentially two independent runs of GBFS, in forward and backward direction, adapted to this setting. Consider first the unidirectional forward variant, i.e., `ChooseDirection` always returns fw. As in GBFS, at each step we expand states with the minimal h-value. The key difference is that, instead of expanding a single state, we expand a set of states, all with the same h-value. Thus, the main loop is iteratively selecting a set of states from open all with the same value, compactly represented as a BDD, and expanding them all at once.

As in previous symbolic search algorithms, we represent transition relations (TRs), the open and closed list for both the forward and backward direction as BDDs. Similarly to GHSETA*, but disregarding operator costs, we construct TRs T_q, classified only by the operator-potential function. We maintain the open and closed list for each direction. The open list contains sets of states ordered by their h-value. The closed list is simply the set of states that have been expanded so far, represented as a single BDD.[1] In the optimal setting, states in the closed list are partitioned by their g-value (i.e., the cost of reaching them). In our case, it suffices to keep a single set of states, disregarding their g and h-value.

In the initialization, the initial state is immediately expanded and thus inserted in the closed list. Then, the set of goal states are expanded and inserted in $\texttt{closed}^{\mathrm{bw}}$, at which point it is checked if the initial state already satisfies the goal. From there on, any set of states generated in the forward search will be intersected with $\texttt{closed}^{\mathrm{bw}}$ (line 17). This

[1] One could use disjunctive partitioning, but we use a single BDD as that worked best in our preliminary experiments.

Algorithm 1: Bidirectional symbolic GBFS with lazy state generation.

Input: A planning task Π, forward and backward operator-potential heuristics $h^{\mathtt{Q}_{\mathrm{fw}}}$ and $h^{\mathtt{Q}_{\mathrm{bw}}}$, parameter $K \in \mathbb{N}$
Output: A plan or "unsolvable".

1 Construct TRs T_q^{fw} and T_q^{bw} for forward and backward direction, respectively, so that each T_q^{x} represents all $o \in \mathcal{O}$ s.t. $\mathtt{Q}_{\mathrm{x}}(o) = q$;
2 $\mathtt{open}^{\mathrm{fw}} \leftarrow \emptyset$; $\mathtt{open}^{\mathrm{bw}} \leftarrow \emptyset$;
3 $\mathtt{closed}^{\mathrm{fw}} \leftarrow \emptyset$; $\mathtt{closed}^{\mathrm{bw}} \leftarrow \emptyset$;
4 ExpandLazy (BDD ($\{I\}$), $h^{\mathtt{Q}_{\mathrm{fw}}}(I)$, fw);
5 **for each** $\langle h, S_G\rangle$ *in partitioning of goal states wrt.* $h^{\mathtt{Q}_{\mathrm{bw}}}$ **do**
6 **if** ExpandLazy *(*S_G*,* h*,* bw*)* **then**
7 **return** empty plan (initial state is a goal state);
8 **while** $\mathtt{open}^{\mathrm{fw}} \neq \emptyset$ *and* $\mathtt{open}^{\mathrm{bw}} \neq \emptyset$ **do**
9 dir $\leftarrow$ ChooseDirection();
10 $h = \min(\{h' \mid \langle h', T, S\rangle \in \mathtt{open}^{\mathrm{dir}}\})$;
11 $S_h \leftarrow$ PopStateSet(h, dir, K);
12 **if** $S_h \neq \emptyset$ **then**
13 **if** ExpandLazy *(*$S_h, h,$ dir*)* **then**
14 **return** ExtractPlan();
15 **return** "unsolvable";
16 **Function** ExpandLazy *(*$S_h, h,$ dir*)***:**
17 **if** $S_h \wedge \mathrm{closed}^{opposite(\mathrm{dir})} \neq \emptyset$ **then**
18 **return** True ;
19 $\mathtt{closed}^{\mathrm{dir}} \leftarrow \mathtt{closed}^{\mathrm{dir}} \vee S_h$;
20 **for each** T_q^{dir} **do**
21 $\mathtt{open}^{\mathrm{dir}} \leftarrow \mathtt{open}^{\mathrm{dir}} \cup \{\langle h+q, T_q^{\mathrm{dir}}, S_h\rangle\}$;
22 **return** False ;
23 **Function** PopStateSet *(*$h,$ dir$, K$*)***:**
24 $S_h \leftarrow \emptyset$;
25 **while** *exists* $\langle h, T, S\rangle$ *in* $\mathtt{open}^{\mathrm{dir}}$ *and* $|S_h| < K$ **do**
26 $S_h \leftarrow S_h \vee (\mathrm{image}^{\mathrm{dir}}(S, T) \wedge \neg\mathtt{closed}^{\mathrm{dir}})$;
27 Remove $\langle h, T, S\rangle$ from $\mathtt{open}^{\mathrm{dir}}$;
28 **return** S_h ;

checks whether a goal state (or any state for which the backward search has already found a plan), has been reached. In that case, we immediately return the plan found. To perform backward search, we treat the set of goal states as the initial state in the forward direction. As states in the open list need to be partitioned by their h-value, we split the set of goal states according to their operator-potential value, following the approach by Fišer, Torralba, and Hoffmann (2024).

3.1 Lazy Image computation

In standard heuristic search algorithms, when a state is expanded all the successors are generated. Most existing symbolic search algorithms follow this and apply the image operation when expanding a set of states. That is, in line 21, they compute $\mathrm{image}^{\mathrm{dir}}(S, T_q^{\mathrm{dir}}) \wedge \neg\mathtt{closed}^{\mathrm{dir}}$, remove duplicates, and insert the set of successors in the open list by computing a disjunction. However, since the bottleneck of the algorithm lies in image computation, computing images to generate sets of states with a high heuristic value that will never be expanded is a waste of resources. Instead, we delay the image computation, duplicate elimination and disjunction to insert in open to the moment where we pop states from the open list, in line 26. In explicit-state heuristic search, a similar idea is applied by Enhanced Partial-Expansion A* (Goldenberg et al. 2014), where to avoid generating and evaluating all successors they re-insert partially expanded nodes in the open list. Here, the bottleneck is always on the image computation and not on the number of elements in the open list, so we can simply consider an open list whose entries correspond to pairs of a set of states and a transition relation (instead of sets of states only). Such a node $\langle h+q, S_h, T_q\rangle$ implicitly represents the set of states that will be generated after computing $\mathrm{image}^{\mathrm{dir}}(S_h, T_q^{\mathrm{dir}})$.

Therefore, expanding a set of states (ExpandLazy) becomes very lightweight. We simply check if any of those states has been reached in the opposite direction, insert the states in the closed list, and insert in open all pairs of the set of states being expanded along with each transition relation. Each such pair $\langle T_q^{\mathrm{dir}}, S_h\rangle$ is implicitly representing the set of states that will be generated when computing $\mathrm{image}^{\mathrm{dir}}(S_h, T_q^{\mathrm{dir}}) \wedge \neg\mathtt{closed}^{\mathrm{dir}}$, and therefore is inserted in open with their h-value $h+q$. Note that ExpandLazy is called on each set of states that is generated by an image computation. Therefore, closed is simultaneously the set of expanded and generated states, and we always perform the goal check with respect to the set of states generated in the opposite direction to stop the search as early as possible.

As a result, the algorithm focuses on computing the image operation only with respect to TRs that decrease the h-value. Consider an example with TRs $\{T_{-1}, T_0, T_1, T_2, T_{10}\}$. This is a typical scenario, as on tasks with unit cost, and a non-zero consistent operator-potential heuristic, there must be a TR T_{-1} representing the operators that decrease the heuristic value, and all other transition relations have a higher value (since, due to consistency in unit-cost tasks the heuristic cannot decrease by more than 1). So, after generating some state set S_h, if it is not empty, it is immediately expanded. In that case, there is always an element $\langle h-1, T_{-1}, S_h\rangle$ inserted in the open list. This is necessarily the element with the lowest heuristic value in the open list (because previously h was the minimum h-value), so the algorithm will pop such a set and compute $\mathrm{image}(S_h, T_{-1})$ to determine if any non-duplicate successor exists. In other words, the algorithm always greedily applies only image computation with respect to operators that decrease the value of the potential heuristic. Only when the result of such an image is empty (i.e., no operator with negative potential is applicable in any state, or it results in a duplicate), other TRs will be considered. But, at that point, we will collect states generated in different ways, e.g. if we have already exhausted all states with the h-value lower than the initial state, we will begin computing images to generate the set of states reachable by sequences of operators with total potential of zero, such as $\langle T_0\rangle$, $\langle T_{-1}, T_1\rangle$, $\langle T_{-1}, T_{-1}, T_2\rangle$, etc.

3.2 Selecting a Set of States for Expansion

At each iteration, the function PopStateSet generates a set of states with minimum heuristic value in order to expand

them all at the same time. A critical question is how many states should be popped in a single iteration. The answer requires considering several conflicting factors.

On the one hand, we desire to represent as many states as possible in order to gain advantage of the symbolic representation. The more states are included in the set, the higher the chances that one of them will be part of a short plan. And it is often more efficient to perform a single image computation for a single larger BDD representing S_h than having to do separate image operations for many subsets of S_h. When considering this, in the extreme case, we would like to generate all states in open with minimum heuristic value and represent them compactly as a single BDD. However, there is no guarantee that such a set of states can be compactly represented and in some cases the algorithm could be completely stalled trying to pop all such states from open.

On the other hand, if we desire to greedily follow the heuristic, there is no need to generate all states with the minimum h value before expanding any of them. In optimal planning, where symbolic search has been traditionally applied, the first criterion is often preferable. The reason is that, except for the last f-layer, all sets of states need to be expanded anyway, as all states with lower f-value than the optimal plan cost must be expanded to prove optimality. However, in satisficing planning it suffices to expand one of those states, if the right one is selected. Therefore, it may be preferable to expand each state as soon as it is generated, hoping to reach states with even lower heuristic values.

The function `PopStateSet` resolves this trade-off by means of a parameter K that controls how many BDD states are generated when extracting a set of states from the open list. Specifically, this is a bound on the size of the BDD (number of BDD nodes) to continue extracting additional sets of states. Setting $K = \infty$ corresponds to always extracting all states with the minimum h-value, and setting $K = 1$ corresponds to always performing a single image operation and immediately returning the set of states, delaying the generation of other states with the same heuristic value. In practice, it is desirable to set K to an intermediate value. This allows extracting more states whenever they can be efficiently represented as a single BDD, while setting a limit on the computation spent on the function for cases where the size of the BDD representation blows up.

3.3 Plan Extraction

To be able to extract a plan, we need to maintain some additional information during the search. For each generated set of states S, let $\mathrm{par}(S)$ denote the parent set of states and let $\mathrm{Tr}(S)$ denote the transition relation that led to S, i.e., $S = \mathrm{image}(\mathrm{par}(S), \mathrm{Tr}(S))$ if S is generated in the forward direction. Since `PopStateSet` can merge multiple BDDs, we also need to maintain which BDDs were merged: if S is the result of merging $S_1, \ldots, S_n$, then we define $\mathrm{merged}(S) = \{S_1, \ldots, S_n\}$; otherwise $\mathrm{merged}(S) = \{S\}$. For the initial state and all merged S (i.e., any S such that $\mathrm{merged}(S) \neq \{S\}$), we leave $\mathrm{par}(S)$ and $\mathrm{Tr}(S)$ undefined.

Whenever a set of states popped from the open list has a non-empty intersection with the closed list of the opposite search direction (line 17), we know that a plan exists: there is a path from the initial state to every state in the intersection, and a path from every state in the intersection to a goal state. To extract one such plan (function `ExtractPlan`), we pick a single "meeting" state from the intersection and reconstruct one concrete path from the initial state to that meeting state and one concrete path from the meeting state to a goal state.

More precisely, assume w.l.o.g. that plan extraction is triggered after a state expansion in the *forward* direction, i.e., we obtained a set of states S from `PopStateSet` (line 11) such that $S \wedge \mathrm{closed}^{\mathrm{bw}} \neq \emptyset$. We then select a set of states (BDD) S^{bw} that was closed in the backward search (and therefore $S^{\mathrm{bw}} \subseteq \mathrm{closed}^{\mathrm{bw}}$) such that $S^{\mathrm{bw}} \wedge S \neq \emptyset$. Finally, we sample a concrete state s from $S^{\mathrm{bw}} \wedge S$ and reconstruct a path from the initial state to s as follows: (a) find $S' \in \mathrm{merged}(S)$ such that $s \in S'$; (b) compute $S'' = \text{pre-image}(s, \mathrm{Tr}(S')) \wedge \mathrm{par}(S')$; (c) sample a concrete state s' from S''; and continue in this manner until we reach the initial state. Similarly, we reconstruct a path from s to a goal state by following the backward search information and using image instead of pre-image. Once we have all concrete states of the plan, we select the operator for each consecutive pair of states in the plan by checking which operator is applicable in the first state and leads to the second state (for which we consider only operators from $\mathrm{Tr}(S)$).

In terms of runtime, plan extraction rarely constitutes a performance bottleneck. In our experiments, plan extraction requires less than $1s$ and accounts for less than 10% of the total runtime in approximately 95% of tasks. Although the underlying operations (BDD intersections, pre-image, and image) can be computationally expensive, they are performed on BDDs representing individual states, which are always compact. Still, in a few domains (visitall, airport, openstacks) there is a noticeable overhead due to plans being long (more than 1000 steps), and the transition relations being large. However, this is often not dominating the total time, except in visitall where it sometimes exceeds 50% of the total runtime.

3.4 Ensemble of Potential Heuristics

We can also run the bi-directional symbolic GBFS with multiple operator-potential heuristic functions. The idea is very simple, instead of each operator-potential being a single numeric value, it corresponds to a vector of values. Suppose we have multiple operator potential functions $\mathtt{Q}_1, \ldots, \mathtt{Q}_n$. Each $\mathtt{Q}_i$ together with the corresponding $h^{\mathtt{P}}_i(I)$ induce the operator potential heuristic. Then, every occurrence of q and h in the algorithm (referring to the operator-potential of a TR and the heuristic value of a set of states, respectively) are replaced by vectors $\vec{q} = \langle q_1, \ldots, q_n \rangle$ and $\vec{h} = \langle h_1, \ldots, h_n \rangle$. Not much changes in the algorithm, we just treat the whole tuple of $\vec{h}$-values as one number. This is similar to how BDDA* and GHSETA* algorithms split the states into sets of states with the same g and h value.

To decide the expansion order, the min expression can be replaced by any function that selects an element $\vec{h}$, out of the different options in the open list. In our experiments, we simply compare the vectors lexicographically, so that

we always select the set of states with minimum heuristic value for the first heuristic, and break ties using the second heuristic. But many other options are possible. For example, if three heuristics are used, one could select the tuple $\vec{h} = \langle h_1, h_2, h_3 \rangle$ minimizing the sum $h_1 + h_2 + h_3$, the maximum $\max(h_1 + h_2 + h_3)$, or any other expression.

One important question is how adding multiple heuristics affects the performance of the algorithm. In explicit-state search, additional heuristics only affect the expansion order of the states, plus some computational overhead of the time spent computing the heuristic itself. However, in symbolic search, it also affects the state and TR partitioning. For the TRs, this is not a problem as by construction each operator has a well-defined operator potential so the number of TRs is bounded by the number of operators. However, only states with the same value for all of the heuristics will be aggregated into a set of states. The main advantage of symbolic search comes from representing large sets of states compactly, so including additional heuristics could be detrimental due to separating too many states into different sets. For example, when initializing the backward search, the algorithm splits all goal states into separate buckets according to their heuristic value. So, this could be problematic if too many heuristics are used and they disagree on the estimate for different goal states.

4 Goal Counting as a Potential Heuristic

So far, all known methods for computing operator-potential heuristics produce admissible heuristics. As this is not necessary for satisficing planning, we introduce a new simple operator-potential heuristic based on the well-known goal counting heuristic that returns, for each state s, the number of goal facts not satisfied in s, i.e., $|G \setminus s|$.

The goal counting operator-potential heuristic is structured similarly to the previous operator-potential heuristics. We associate each operator with a change of the heuristic value, using an operator-potential function $\mathtt{Q}_{\mathrm{gc}}$. Then the heuristic value for a state s reached in the forward direction by a sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ is computed as $|G \setminus I| + \sum_{i=1}^{n} \mathtt{Q}_{\mathrm{gc}}(o_i)$, i.e., the heuristic value of the goal counting heuristic for the initial state plus the sum of operator-potentials over the sequence π. For the backward direction, we use the same expression $|G \setminus I| + \sum_{i=1}^{n} \mathtt{Q}_{\mathrm{gc}}(o_i)$, this time over s-plans $\pi = \langle o_1, \ldots, o_n \rangle$. Since the heuristic values for both search directions are defined in the same way, we formally introduce only one heuristic function $h^{\mathtt{Q}_{\mathrm{gc}}}$ defined over sequences of operators. Formally, the goal counting operator-potential *forward* heuristic is $h^{\mathtt{Q}_{\mathrm{gc}}}$ restricted to I-s-paths, and the goal counting operator-potential *backward* heuristic is $h^{\mathtt{Q}_{\mathrm{gc}}}$ restricted to s-plans.

Definition 1. Given a planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$, the **goal counting operator-potential function** $\mathtt{Q}_{\mathrm{gc}} \colon \mathcal{O} \to \mathbb{N}_0$ is defined as $\mathtt{Q}_{\mathrm{gc}}(o) = |\mathrm{pre}_o \cap G| - |\mathrm{eff}_o \cap G|$ for every operator $o \in \mathcal{O}$.

Given the goal counting operator-potential function $\mathtt{Q}_{\mathrm{gc}}$, the **goal counting operator-potential heuristic** is defined as $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) = |G \setminus I| + \sum_{i=1}^{n} \mathtt{Q}_{\mathrm{gc}}(o_i)$ for every sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$.

Now we show that in the general case, $h^{\mathtt{Q}_{\mathrm{gc}}}$ in the forward direction (Proposition 3) is a lower bound on the goal counting heuristic, i.e., it is at most $|G \setminus s|$ for all forward reachable states s. In the backward direction (Proposition 4), $h^{\mathtt{Q}_{\mathrm{gc}}}$ is a lower bound on $|G \setminus I| - |G \setminus s|$ for all backward reachable states s, i.e., on the signed difference between the goal counting heuristic estimate for the initial state and s.

Lemma 2. *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ be a planning task, let $\mathtt{Q}_{\mathrm{gc}}$ be the goal counting operator-potential function, let s be a state, and let $o \in \mathcal{O}$ be an operator applicable in s. Then $\mathtt{Q}_{\mathrm{gc}}(o) \leq |G \setminus o[\![s]\!]| - |G \setminus s|$.*

Proof. Let $D = (G \cap s) \setminus o[\![s]\!]$ be the set of goal facts deleted by o, and let $A = (G \cap o[\![s]\!]) \setminus s$ be the set of goal facts added by o. Clearly, $|G \setminus o[\![s]\!]| = |G \setminus s| + |D| - |A|$, and therefore we need to show that $\mathtt{Q}_{\mathrm{gc}}(o) \leq |D| - |A|$. Since facts A are added by o, it follows that $A \subseteq \mathrm{eff}_o \cap G$ and therefore $|A| \leq |\mathrm{eff}_o \cap G|$. Since facts D are deleted by o, it follows that for all $\langle V, v \rangle \in D$ there is $\langle V, v' \rangle \in \mathrm{eff}_o$ s.t. $v \neq v'$. And since for every $\langle V, v \rangle \in \mathrm{pre}_o$ we have that $\langle V, v' \rangle \in \mathrm{eff}_o$, $v \neq v'$, by definition, it follows that $\mathrm{pre}_o \cap G \subseteq D$ and therefore $|\mathrm{pre}_o \cap G| \leq |D|$. Therefore, we have that $\mathtt{Q}_{\mathrm{gc}}(o) = |\mathrm{pre}_o \cap G| - |\mathrm{eff}_o \cap G| \leq |D| - |A|$, which concludes the proof. □

Proposition 3. *Let Π and $\mathtt{Q}_{\mathrm{gc}}$ be as before. Then for every I-s-path π it holds that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) \leq |G \setminus s|$.*

Proof. (By induction) For the initial state, we have that $h^{\mathtt{Q}_{\mathrm{gc}}}(\langle\rangle) = |G \setminus I|$. Let π be an I-s-path, and let $o \in \mathcal{O}$ be an operator applicable in s. We assume that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) \leq |G \setminus s|$, and we need to show that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + \mathtt{Q}_{\mathrm{gc}}(o) \leq |G \setminus o[\![s]\!]|$. From Lemma 2 it follows that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + \mathtt{Q}_{\mathrm{gc}}(o) \leq h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + |G \setminus o[\![s]\!]| - |G \setminus s|$. From the assumption we have that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + |G \setminus o[\![s]\!]| - |G \setminus s| \leq |G \setminus s| + |G \setminus o[\![s]\!]| - |G \setminus s| = |G \setminus o[\![s]\!]|$, which concludes the proof. □

Proposition 4. *Let Π and $\mathtt{Q}_{\mathrm{gc}}$ be as before. Then for every s-plan π it holds that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) \leq |G \setminus I| - |G \setminus s|$.*

Proof. (By induction) For any goal state s_g, we have that $h^{\mathtt{Q}_{\mathrm{gc}}}(\langle\rangle) = |G \setminus I| = |G \setminus I| - |G \setminus s_g|$ as $G \subseteq s_g$. We assume that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) \leq |G \setminus I| - |G \setminus s|$ for some s-plan π, and we need to show that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + \mathtt{Q}_{\mathrm{gc}}(o) \leq |G \setminus I| - |G \setminus s'|$ for any state s' and operator $o \in \mathcal{O}$ such that o is applicable in s' and $o[\![s']\!] = s$. From Lemma 2 it follows that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + \mathtt{Q}_{\mathrm{gc}}(o) \leq h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + |G \setminus s| - |G \setminus s'|$, and from the assumption it follows that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + |G \setminus s| \leq |G \setminus I|$. Therefore, we have that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) + |G \setminus s| - |G \setminus s'| \leq |G \setminus I| - |G \setminus s'|$, which concludes the proof. □

In normalized planning tasks (which we use in our experiments), the goal counting operator-potential heuristic does not return lower bounds, but it is exactly equal to $|G \setminus s|$ in the forward direction and to $|G \setminus I| - |G \setminus s|$ in the backward direction. In other words, in normalized tasks, $h^{\mathtt{Q}_{\mathrm{gc}}}$ is state-dependent in both search directions as the values are the same for all I-s-paths and s-plans, respectively.

Proposition 5. *Let Π and $\mathtt{Q}_{\mathrm{gc}}$ be as before. If Π is normalized, then*

(1) for every I-s-path π it holds that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) = |G \setminus s|$;
(2) for every s-plan π it holds that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) = |G \setminus I| - |G \setminus s|$;
(3) for every two distinct I-s-paths or s-plans π and π', it holds that $h^{\mathtt{Q}_{\mathrm{gc}}}(\pi) = h^{\mathtt{Q}_{\mathrm{gc}}}(\pi')$, i.e., $h^{\mathtt{Q}_{\mathrm{gc}}}$ is state-dependent in both search directions.

Proof. Let s be a state, and let $o \in \mathcal{O}$ be an operator applicable in s. Since Π is normalized, then $\mathcal{V}(\mathrm{pre}_o) = \mathcal{V}(\mathrm{eff}_o)$ and $\mathrm{pre}_o \cap \mathrm{eff}_o = \emptyset$. Therefore, the set of goal facts deleted by o is exactly $\mathrm{pre}_o \cap G = (G \cap s) \setminus o[\![s]\!]$, and the set of goal facts added by o is exactly $\mathrm{eff}_o \cap G = (G \cap o[\![s]\!]) \setminus s$. Therefore, analogously to the proof of Lemma 2, we have that $\mathtt{Q}_{\mathrm{gc}}(o) = |\mathrm{pre}_o \cap G| - |\mathrm{eff}_o \cap G| = |G \setminus o[\![s]\!]| - |G \setminus s|$.

So, (1) follows analogously to the proof of Proposition 3, and (2) analogously to the proof of Proposition 4 where we use the fact that $\mathtt{Q}_{\mathrm{gc}}(o) = |G \setminus o[\![s]\!]| - |G \setminus s|$ for any state s and $o \in \mathcal{O}$ applicable in s instead of $\mathtt{Q}_{\mathrm{gc}}(o) \leq |G \setminus o[\![s]\!]| - |G \setminus s|$. (3) follows directly from (1) and (2). □

Proposition 5 shows that in normalized tasks, the goal counting operator-potential heuristic is exactly equal to the goal counting heuristic in the forward direction. In the backward direction, $h^{\mathtt{Q}_{\mathrm{gc}}}$ measures how many goal facts need to be removed to reach the initial state. However, the difference $|G \setminus I| - |G \setminus s|$ is signed which means that if some goal facts are already achieved in I, the heuristic will prefer states that have even less goal facts achieved over those that have more goal facts achieved. Nevertheless, in most tasks, no goal facts are satisfied in the initial state.

Note also that the goal counting heuristic can be formulated as a potential heuristic where we set potentials to 1 for all facts from goal variables that are not satisfied in the goal and to 0 for all other facts. Let $\mathtt{P}_{\mathrm{gc}} \colon \mathcal{F} \to \mathbb{R}$ be a potential function defined as $\mathtt{P}_{\mathrm{gc}}(\langle V, v \rangle) = 1$ for all facts $\langle V, v \rangle$ such that $V \in \mathcal{V}(G)$, $v \in \mathrm{dom}(V)$ and $\langle V, v \rangle \notin G$, and $\mathtt{P}_{\mathrm{gc}}(f) = 0$ for every other fact f. The goal counting potential heuristic $h^{\mathtt{P}_{\mathrm{gc}}}$ is then defined as $h^{\mathtt{P}_{\mathrm{gc}}}(s) = \sum_{f \in s} \mathtt{P}_{\mathrm{gc}}(f)$ for every state s. Based on $h^{\mathtt{P}_{\mathrm{gc}}}$, we can construct the goal counting operator-potential heuristic $h^{\mathtt{Q}_{\mathrm{gc}}}$ following the approach of Fišer, Torralba, and Hoffmann (2024).

5 Experimental Evaluation

We implemented our algorithms in C as part of the cpddl planning system (Fišer 2025; Fišer and Torralba 2026). We used all planning domains from the satisficing track of the International Planning Competitions (IPCs) from 1998 to 2023 excluding the ones containing conditional effects after translation. We merged, for each domain, all benchmark suites across different IPCs, resulting in 56 domains and 1 990 tasks overall. We used a cluster of computing nodes with AMD EPYC 7642 processors and CPLEX (I)LP solver v22.11.1 for computing operator-potential heuristics. The time and memory limits were set to 30 minutes and 8 GB, respectively.

Besides the goal counting operator-potential heuristic (gc) described in Section 4, we used the following (admissible) operator-potential heuristic from prior work (Fišer, Torralba, and Hoffmann 2024), computed using CPLEX MIP solver v22.11.1: AI that maximizes the heuristic value for the average syntactic state while enforcing the maximum heuristic value for the initial state (Seipp, Pommerening, and Helmert 2015; Fišer, Horčík, and Komenda 2020). We experimented with several different operator-potential heuristics, but using AI always resulted in a higher coverage.

Operators and facts are pruned with the h^2 preprocessor in the forward and backward direction (Alcázar and Torralba 2015), and the translation from PDDL to FDR uses the inference of mutex groups proposed by Fišer (2020). We used a time limit of 10 seconds for merging transition relation BDDs and 30 seconds for applying mutexes on the goal BDD. In case of bi-directional search, if mutexes cannot be applied on the goal BDD within the time limit, the backward search is disabled which occurred in 237 tasks. When merging transition relation BDDs, we never attempt to merge BDDs with more than 10 000 nodes. All tasks were normalized by the "multiplication" method described in Section 2—it was as beneficial here as previously observed in the context of optimal planning (Fišer, Torralba, and Hoffmann 2024).

We evaluated several combinations of operator-potential heuristics in forward and backward direction of the symbolic GBFS. Here, we report the three best performing variants of forward-only and bi-directional search (backward-only search was clearly inferior, so we omit it). Moreover, we add the forward-only search with gc as it is a newly introduced heuristic. The compared symbolic GBFS configurations are thus the following. $\overrightarrow{\texttt{AI}}$, $\overrightarrow{\texttt{gc}}$: forward-only search with AI and gc, respectively; $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$, $\overrightarrow{\texttt{gc}\succ\texttt{AI}}$: forward-only search with open lists ordered first by AI and then by gc, and vice-versa, respectively; $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$-$\overleftarrow{\texttt{b}}$, $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$-$\overleftarrow{\texttt{gc}}$, $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$-$\overleftarrow{\texttt{AI}\succ\texttt{gc}}$: bi-directional search using AI$\succ$gc for the forward direction, and blind search, gc, and AI$\succ$gc in the backward direction, respectively. As baselines, we use the LAMA planner (Richter and Westphal 2010), denoted as lama, and several variants of explicit-state heuristic GBFS methods: the GBFS with the goal counting heuristic (gbfs-gc), the AI variant of the potential heuristic (gbfs-AI), and the combination of the two heuristics (gbfs-AI$\succ$gc) where states are ordered first by AI and then by the goal counting heuristic. We also use the lazy variant of GBFS (Richter and Helmert 2009) with the FF heuristic (Hoffmann and Nebel 2001), denoted as lazy-ff. The LAMA planner is used without any modifications in the version used in IPC'23. For the GBFS methods, we apply the same h^2 pruning as used in our symbolic GBFS method—it increases their coverage and makes the comparison easier to interpret. As a symbolic search baseline, we use the best-performing variant of a symbolic optimal planner (smb-AI): bi-directional A* with AI in the forward direction, and the blind search in the backward direction. All planners except lama were run on tasks transformed to unit cost as it led to a higher number of solved tasks.

In bi-directional search, the backward direction is disabled whenever a backward step takes longer than 3 minutes to allow progress at least in the forward direction. This occurred in 44, 74, and 57 tasks for $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$-$\overleftarrow{\texttt{b}}$, $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$-$\overleftarrow{\texttt{gc}}$, and $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$-$\overleftarrow{\texttt{AI}\succ\texttt{gc}}$, respectively (with K=1 000 000, see below). The average number of backward steps before dis-

	Domain Dominance													Task Dominance														
	lama	$\overrightarrow{\texttt{AI}\succ\texttt{gc}}$	$\overleftarrow{\texttt{AI}\succ\texttt{gc-b}}$	gbfs-AI≻gc	lazy-ff	$\overrightarrow{\texttt{gc}\succ\texttt{AI}}$	$\overleftarrow{\texttt{AI}\succ\texttt{gc-gc}}$	$\overleftarrow{\texttt{AI}\succ\texttt{gc-AI}\succ\texttt{gc}}$	$\overrightarrow{\texttt{AI}}$	$\overrightarrow{\texttt{gc}}$	gbfs-AI	gbfs-gc	smb-AI	lama	$\overrightarrow{\texttt{AI}\succ\texttt{gc}}$	$\overleftarrow{\texttt{AI}\succ\texttt{gc-b}}$	gbfs-AI≻gc	lazy-ff	$\overrightarrow{\texttt{gc}\succ\texttt{AI}}$	$\overleftarrow{\texttt{AI}\succ\texttt{gc-gc}}$	$\overleftarrow{\texttt{AI}\succ\texttt{gc-AI}\succ\texttt{gc}}$	$\overrightarrow{\texttt{AI}}$	$\overrightarrow{\texttt{gc}}$	gbfs-AI	gbfs-gc	smb-AI	tot	rel-tot
lama	–	**29**	**30**	**31**	**33**	**30**	**34**	**31**	**28**	**33**	**34**	**33**	**43**	–	**265**	**282**	**278**	**301**	**282**	**312**	**332**	**359**	**367**	**391**	**423**	**793**	1 713	44.6
$\overrightarrow{\texttt{AI}\succ\texttt{gc}}$	13	–	**10**	**22**	**23**	**13**	**22**	**23**	11	**23**	**24**	**24**	**41**	63	–	**23**	**139**	**178**	**52**	**56**	**82**	**120**	**161**	**228**	**294**	**598**	1 511	38.4
$\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{b}}}$	14	5	–	**20**	20	**16**	**20**	**19**	13	**23**	**22**	**24**	**40**	77	20	–	**152**	**170**	**70**	**57**	**83**	**133**	**170**	**236**	**297**	**584**	1 508	38.2
gbfs-AI≻gc	11	15	19	–	**23**	17	**25**	**24**	**20**	**23**	**15**	**19**	**39**	70	133	149	–	**193**	**156**	**177**	**204**	**211**	**236**	**130**	**230**	**610**	1 505	39.2
lazy-ff	6	19	**21**	22	–	**23**	**22**	**21**	**22**	**28**	**23**	**25**	**43**	77	156	151	177	–	**176**	**176**	**185**	**239**	**221**	**259**	**286**	**558**	1 489	38.4
$\overrightarrow{\texttt{gc}\succ\texttt{AI}}$	13	4	13	**18**	21	–	**21**	**22**	10	**18**	**22**	**23**	**38**	53	25	46	135	171	–	**71**	**97**	**118**	**118**	**225**	**270**	**574**	1 484	37.7
$\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{gc}}}$	11	2	9	17	18	11	–	8	9	**20**	20	**24**	**38**	80	26	30	153	168	68	–	**52**	**124**	**157**	**239**	**287**	**568**	1 481	37.3
$\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{AI}\succ\texttt{gc}}}$	11	4	10	17	19	10	**9**	–	9	**21**	19	**25**	**37**	86	38	42	166	163	80	38	–	**140**	**163**	**251**	**298**	**574**	1 467	37.1
$\overrightarrow{\texttt{AI}}$	13	11	**17**	19	20	**17**	**23**	**25**	–	**22**	**22**	**26**	**38**	65	28	44	125	169	53	62	92	–	**138**	**200**	**250**	**501**	1 419	37.1
$\overrightarrow{\texttt{gc}}$	11	10	13	18	15	10	16	15	10	–	20	**18**	**34**	56	52	64	133	134	36	78	98	121	–	**216**	**183**	**489**	1 402	35.3
gbfs-AI	11	13	17	6	19	16	**21**	**21**	18	**23**	–	17	**37**	69	108	119	16	161	132	149	175	172	205	–	**213**	**512**	1 391	36.8
gbfs-gc	9	14	15	13	19	16	18	17	14	17	**22**	–	**37**	57	130	136	72	144	133	153	178	178	128	169	–	**475**	1 347	35.0
smb-AI	7	6	6	8	5	8	8	8	6	9	10	11	–	36	43	32	61	25	46	43	63	38	43	77	84	–	956	24.0

Table 2: "Domain Dominance": the number of domains where the row method solved more tasks than the column method; "Task Dominance": the number of tasks solved by the row method that were not solved by the column method; the number at (x, y) is shown bold if higher than the number at (y, x); "tot": total number of solved tasks; "rel-tot": the overall number of solved tasks in relative terms: sum of S_d/N_d over all domains d where S_d is the number of solved tasks and N_d is the overall number of tasks in the domain.

	parameter K						
	1	1K	10K	100K	1M	10M	100M
$\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{b}}}$	1 445	1 470	1 499	1 494	**1 508**	1 502	1 507
$\overrightarrow{\texttt{AI}\succ\texttt{gc}}$	1 451	1 474	1 509	1 510	**1 511**	**1 511**	1 505
$\overrightarrow{\texttt{gc}}$	1 323	1 340	1 361	1 376	**1 402**	1 393	1 392
$\overrightarrow{\texttt{gc}\succ\texttt{AI}}$	1 404	1 425	1 453	1 468	**1 484**	1 479	1 478
$\overrightarrow{\texttt{AI}}$	1 433	1 452	**1 483**	1 480	1 419	1 399	1 399
$\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{AI}\succ\texttt{gc}}}$	1 434	1 450	**1 485**	1 469	1 467	1 459	1 452
$\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{gc}}}$	1 465	1 488	**1 506**	1 475	1 481	1 485	1 477

Table 1: Number of solved tasks for different values of the parameter K going from 1 (no state merging) to 100 million.

abling the backward search was 3.5, 15.2, and 13.4 for the three variants, respectively. So, the backward search is rarely disabled and usually after only a few steps, but it increases coverage.

Table 1 shows the overall number of solved tasks (coverage) for different values of the parameter K that controls merging of state BDDs. $K = 1$ means no merging is performed, and the higher the value of K, the more BDDs are merged. Merging seems to be universally beneficial, but there are two different sweet spots, $K = 10\,000$ and $K = 1\,000\,000$, depending on the used heuristics. We also observed variations between domains, for example, the drop in coverage for $\overrightarrow{\texttt{AI}}$ from 10K to 1M is mainly due to the openstacks domain where the coverage drops by 45. So, we think the performance could be improved with some dynamic strategy for choosing K. For the rest of the experiments, we use $K = 1\,000\,000$ as it leads to the highest coverage of the best variant of symbolic GBFS.

Table 2 shows the number of domains where one method (row) solved more tasks than the other (column), the number of tasks solved by one method (row) but not the other (column), and the coverage in absolute terms and relative terms as the sum of fractions of solved tasks per domain.[2]

Although lama is the best-performing method overall, the symbolic GBFS variants are competitive with all other explicit-state search baselines: $\overrightarrow{\texttt{gc}}$, $\overrightarrow{\texttt{AI}}$, and $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ solve more tasks than gbfs-gc, gbfs-AI, and gbfs-AI≻gc, respectively. Two of our configurations beat lazy-ff with two more solving only slightly fewer tasks. Moreover, $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ is the second best-performing method after lama in the overall coverage. The symbolic baseline smb-AI is clearly inferior, showing that the performance of symbolic GBFS is not merely due to representing whole sets of states as BDDs but also due to the use of operator-potential heuristics in a greedy manner.

Figure 1 shows the runtime comparisons between the symbolic GBFS variants and their explicit-state search baselines, and between the best-performing symbolic GBFS variant, and lazy-ff and lama. The comparison shows that the symbolic GBFS methods are often slower than their explicit-state counterparts on commonly solved tasks, which is not surprising given the overhead of symbolic search. Nevertheless, $\overrightarrow{\texttt{gc}}$ is faster than gbfs-gc in 155 commonly solved tasks, $\overrightarrow{\texttt{AI}}$ is faster than gbfs-AI in 140 com-

[2]The coverage with the original cost was significantly lower for all methods except for gbfs-gc where the coverage did not change. lazy-ff: 1 349; gbfs-AI≻gc: 1 461; gbfs-AI: 1 248; gbfs-gc: 1 347; $\overrightarrow{\texttt{AI}}$: 1 320; $\overrightarrow{\texttt{gc}\succ\texttt{AI}}$: 1 467; $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$: 1 462; $\overrightarrow{\texttt{gc}}$: 1 393; $\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{b}}}$: 1 478; $\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{AI}\succ\texttt{gc}}}$: 1 446; $\overrightarrow{\texttt{AI}\succ\texttt{gc-}\overleftarrow{\texttt{gc}}}$: 1 461.

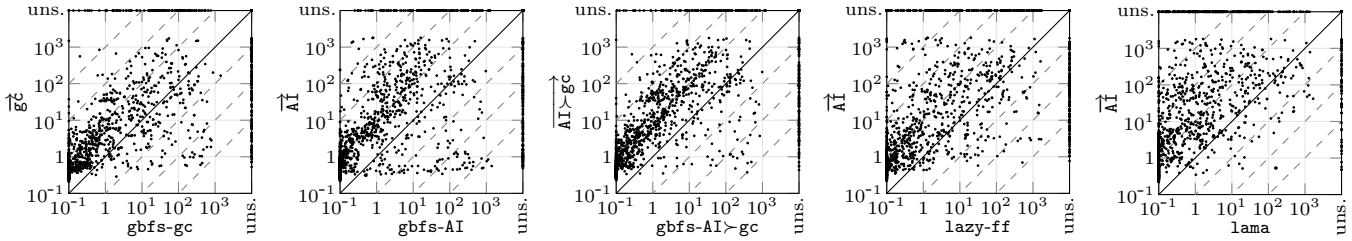

Figure 1: The comparison of runtime (in seconds). For lama, we take the time of the first solution found.

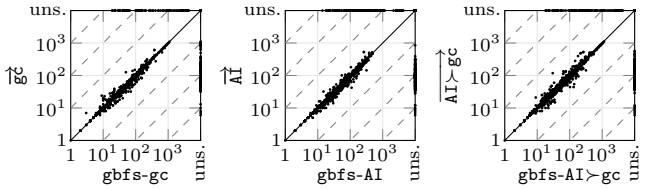

Figure 2: The comparison of plan lengths.

monly solved tasks, and $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ is faster than gbfs-AI≻gc, lazy-ff, and lama in 95, 167, and 108 commonly solved tasks, respectively.

In terms of plan length, there is not a very significant difference between the symbolic and explicit GBFS with the same heuristics (also see Figure 2). The average and median plan lengths on commonly solved tasks are slightly in favor of the symbolic GBFS variants. Specifically, $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ vs. gbfs-AI≻gc: 92.7 vs. 95.4 (average), 50 vs. 53 (median); $\overrightarrow{\texttt{gc}}$ vs. gbfs-gc: 81.7 vs. 83.2 (average), 43 vs. 45 (median); $\overrightarrow{\texttt{AI}}$ vs. gbfs-AI: 65.5 vs. 68.5 (average), 41 vs. 44 (median).

In case of $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ vs. lama, lama produces shorter plans more often than the other way around (average 93.1 vs. 87.5 and median 52 vs. 44), but this is due to the fact that lama keeps improving solutions until the time limit. When we compared $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ with the first plans found by lama, the results slightly favor $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ (average 93.1 vs. 97.3 and median 52 vs. 52).

Table 2 also clearly shows the complementarity of the symbolic and explicit-state search methods. For example, $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ solves more tasks than gbfs-AI≻gc, lazy-ff, gbfs-AI and gbfs-gc in 22, 23, 24 and 24 domains, respectively, whereas gbfs-AI≻gc, lazy-ff, gbfs-AI and gbfs-gc solve more tasks than $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ in 15, 19, 13 and 14 domains, respectively. Similarly, $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ solves between 139 and 294 tasks that the gbfs-/lazy- baselines could not solve, whereas those baselines solve between 108 and 156 tasks that $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ could not solve.

When looking more closely at the domains where one method dominates the other, we observe that in most cases either the symbolic GBFS *strictly dominates* the explicit-state search method in the sense that it solves all tasks solved by the explicit-state search method and more, or the the explicit-state search method strictly dominates the symbolic GBFS counterpart. Namely, out of 18 domains where $\overrightarrow{\texttt{gc}}$ solves more tasks than gbfs-gc, there are 17 domains where $\overrightarrow{\texttt{gc}}$ solves all tasks solved by gbfs-gc. Conversely, out of 17 domains where gbfs-gc dominates $\overrightarrow{\texttt{gc}}$, there are 15 domains where gbfs-gc strictly dominates $\overrightarrow{\texttt{gc}}$. And similarly for other variants: $\overrightarrow{\texttt{AI}}$ strictly dominates (dominates) gbfs-AI in 19 (22) domains; gbfs-AI strictly dominates (dominates) $\overrightarrow{\texttt{AI}}$ in 13 (18) domains; $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$ vs. gbfs-AI≻gc: 18 (22) domains; and gbfs-AI≻gc vs. $\overrightarrow{\texttt{AI}\succ\texttt{gc}}$: 12 (15) domains. This indicates that the observed performance differences are not simply due to arbitrary tie-breaking decisions, but rather stem from how effectively the underlying BDD representation captures the sets of states encountered during the search.

When comparing the symbolic GBFS variants with lama, we can see that lama has significantly higher coverage overall, but there are still many domains where using our method pays off, indicating the symbolic GBFS with operator-potential heuristics is truly complementary to the state-of-the-art explicit-state search methods and is worth considering as a part of planning portfolios.

6 Conclusion

In this paper, we have considered symbolic GBFS with admissible operator-potential heuristics for satisficing planning. We showed that these approaches that were traditionally considered only suitable for optimal planning, are a viable alternative for satisficing planning, with complementary strengths to other state-of-the-art approaches. This opens many avenues of research for future work: can some of the search enhancements for satisficing planning such as landmark heuristics (Hoffmann, Porteous, and Sebastia 2004), multiple queues (Röger and Helmert 2010; Corrêa and Seipp 2025), or novelty pruning (Lipovetzky and Geffner 2017; Katz et al. 2017; Fickert and Hoffmann 2022) be applied as well in symbolic search? Are there other ways of generating potential heuristics that are tailored for satisficing search? So far, few approaches have been considered (Francès et al. 2019; Helmert et al. 2022) and no automatic alternative exists.

References

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 2–6.

Behnke, G.; and Speck, D. 2021. Symbolic Search for Optimal Total-Order HTN Planning. In *Proceedings of the*

35th AAAI Conference on Artificial Intelligence (AAAI'21), 11744–11754.

Bonet, B.; and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence*, 129(1–2): 5–33.

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

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147(1–2): 35–84.

Corrêa, A. B.; and Seipp, J. 2025. Alternation-Based Novelty Search. In *Proceedings of the 35th International Conference on Automated Planning and Scheduling (ICAPS'25)*.

Edelkamp, S.; and Helmert, M. 2001. MIPS: The Model Checking Integrated Planning System. *AI Magazine*, 22(3): 67–71.

Edelkamp, S.; and Kissmann, P. 2008. Limits and Possibilities of BDDs in State Space Search. In *Proceedings of the 23rd National Conference of the American Association for Artificial Intelligence (AAAI'08)*, 1452–1453.

Edelkamp, S.; and Kissmann, P. 2009. Optimal Symbolic Planning with Action Costs and Preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, 1690–1695.

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2015. BDDs Strike Back (in AI Planning). In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, 4320–4321.

Edelkamp, S.; and Reffel, F. 1998. OBDDs in Heuristic Search. In *Proceedings of the 22nd Annual German Conference on Advances in Artificial Intelligence (KI'98)*, volume 1504 of *Lecture Notes in Computer Science*, 81–92.

Fickert, M.; and Hoffmann, J. 2022. Online Relaxation Refinement for Satisficing Planning: On Partial Delete Relaxation, Complete Hill-Climbing, and Novelty Pruning. *Journal of Artificial Intelligence Research*, 73: 67–115.

Fišer, D. 2025. cpddl. Zenodo. https://doi.org/10.5281/zenodo.16423301.

Fišer, D. 2020. Lifted Fact-Alternating Mutex Groups and Pruned Grounding of Classical Planning Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*, 9835–9842.

Fišer, D.; Horčík, R.; and Komenda, A. 2020. Strengthening Potential Heuristics with Mutexes and Disambiguations. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20)*, 124–133.

Fišer, D.; and Torralba, Á. 2026. Code and Data for "Symbolic Greedy Best-First Search with Operator-Potential Heuristics". Zenodo. https://doi.org/10.5281/zenodo.20545270.

Fišer, D.; Torralba, Á.; and Hoffmann, J. 2022a. Operator-Potential Heuristics for Symbolic Search. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI'22)*, 9750–9757.

Fišer, D.; Torralba, Á.; and Hoffmann, J. 2022b. Operator-Potentials in Symbolic Search: From Forward to Bidirectional Search. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, 80–89.

Fišer, D.; Torralba, Á.; and Hoffmann, J. 2024. Boosting Optimal Symbolic Planning: Operator-Potential Heuristics. *Artificial Intelligence*, 104174.

Francès, G.; Corrêa, A. B.; Geissmann, C.; and Pommerening, F. 2019. Generalized Potential Heuristics for Classical Planning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI'19)*, 5554–5561.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced Partial Expansion A*. *Journal of Artificial Intelligence Research*, 50: 141–187.

Hansen, E. A.; Zhou, R.; and Feng, Z. 2002. Symbolic Heuristic Search Using Decision Diagrams. In *Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, Proceedings*, volume 2371 of *Lecture Notes in Computer Science*, 83–98.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Helmert, M.; Sievers, S.; Rovner, A.; and Corrêa, A. B. 2022. On the Complexity of Heuristic Synthesis for Satisficing Classical Planning: Potential Heuristics and Beyond. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22)*, 124–133.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research*, 22: 215–278.

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

Katz, M.; Lipovetzky, N.; Moshkovich, D.; and Tuisov, A. 2017. Adapting Novelty to Classical Planning as Heuristic Search. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, 172–180. AAAI Press.

Kissmann, P.; and Edelkamp, S. 2010. Layer-Abstraction for Symbolically Solving General Two-Player Games. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search (SOCS'10)*.

Kissmann, P.; and Edelkamp, S. 2011. Improving Cost-Optimal Domain-Independent Symbolic Planning. In *Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11)*, 992–997.

Lipovetzky, N.; and Geffner, H. 2017. A Polynomial Planning Algorithm that Beats LAMA and FF. In *Proceedings*

of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17), 195–199. AAAI Press.

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

Pommerening, F.; and Helmert, M. 2015. A Normal Form for Classical Planning Tasks. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 188–192.

Pommerening, F.; Helmert, M.; and Bonet, B. 2017. Higher-Dimensional Potential Heuristics for Optimal Classical Planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, 3636–3643.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, 3335–3341.

Pozanco, A.; Torralba, Á.; and Borrajo, D. 2024. Computing Planning Centroids and Minimum Covering States Using Symbolic Bidirectional Search. In *Proceedings of the 34th International Conference on Automated Planning and Scheduling (ICAPS'24)*, 455–463.

Richter, S.; and Helmert, M. 2009. Preferred Operators and Deferred Evaluation in Satisficing Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, 273–280.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Röger, G.; and Helmert, M. 2010. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, 246–249.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New Optimization Functions for Potential Heuristics. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*, 193–201.

Speck, D.; Geißer, F.; and Mattmüller, R. 2020. When Perfect Is Not Good Enough: On the Search Behaviour of Symbolic Heuristic Search. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20)*, 263–271.

Speck, D.; and Helmert, M. 2025. On Performance Guarantees for Symbolic Search in Classical Planning. In *Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025)*, 4628–4636. IOS Press.

Speck, D.; and Katz, M. 2021. Symbolic Search for Oversubscription Planning. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21)*, 11972–11980.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*, 9967–9974.

Speck, D.; Seipp, J.; and Torralba, Á. 2025. Symbolic Search for Cost-Optimal Planning with Expressive Model Extensions. *Journal of Artificial Intelligence Research*, 82: 1349–1405.

Torralba, Á. 2016. SymPA: Symbolic Perimeter Abstractions for Proving Unsolvability. In *UIPC 2016 planner abstracts*, 8–11.

Torralba, Á.; and Alcázar, V. 2013. Constrained Symbolic Search: On Mutexes, BDD Minimization and More. In *Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13)*, 175–183.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A Symbolic Bidirectional A* Planner. In *IPC 2014 planner abstracts*, 105–109.

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

Torralba, Á.; Edelkamp, S.; and Kissmann, P. 2013. Transition Trees for Cost-Optimal Symbolic Planning. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 206–214.

Torralba, Á.; Gnad, D.; Dubbert, P.; and Hoffmann, J. 2016. On State-Dominance Criteria in Fork-Decoupled Search. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*.

Torralba, Á.; Speicher, P.; Künnemann, R.; Steinmetz, M.; and Hoffmann, J. 2021. Faster Stackelberg Planning via Symbolic Search and Information Sharing. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21)*, 11998–12006.