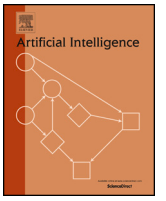
Artificial Intelligence 334 (2024) 104174

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Boosting optimal symbolic planning: Operator-potential heuristics ☆

Daniel Fišer [a,*], Álvaro Torralba [b], Jörg Hoffmann [a,c]

[a] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[b] Aalborg University, Denmark
[c] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

ARTICLE INFO

Keywords:
Classical planning
Heuristic search
Symbolic search
Potential heuristics

ABSTRACT

Heuristic search guides the exploration of states via heuristic functions h estimating remaining cost. Symbolic search instead replaces the exploration of individual states with that of state sets, compactly represented using binary decision diagrams (BDDs). In cost-optimal planning, heuristic explicit search performs best overall, but symbolic search performs best in many individual domains, so both approaches together constitute the state of the art. Yet combinations of the two have so far not been an unqualified success, because (i) h must be applicable to sets of states rather than individual ones, and (ii) the different state partitioning induced by h may be detrimental for BDD size. Many competitive heuristic functions in planning do not qualify for (i), and it has been shown that even extremely informed heuristics can deteriorate search performance due to (ii).

Here we show how to achieve (i) for a state-of-the-art family of heuristic functions, namely potential heuristics. These assign a fixed potential value to each state-variable/value pair, ensuring by LP constraints that the sum over these values, for any state, yields an admissible and consistent heuristic function. Our key observation is that we can express potential heuristics through fixed potential values for operators instead, capturing the change of heuristic value induced by each operator. These reformulated heuristics satisfy (i) because we can express the heuristic value change as part of the BDD transition relation in symbolic search steps. We run exhaustive experiments on IPC benchmarks, evaluating several different instantiations of potential heuristics in forward, backward, and bi-directional symbolic search. Our operator-potential heuristics turn out to be highly beneficial, in particular they hardly ever suffer from (ii). Our best configurations soundly beat previous optimal symbolic planning algorithms, bringing them on par with the state of the art in optimal heuristic explicit search planning in overall performance.

1. Introduction

Classical planning deals with problems of finding a sequence of operators (or actions) leading from an initial state to one of the goal states in a fully observable deterministic environment. In this paper, we are concerned with two families of methods designed

☆ This paper is an invited revision of a paper which first appeared at the 2022 International Conference of the Association for the Advancement of Artificial Intelligence (AAAI-22).

* Corresponding author.
E-mail addresses: danfis@danfis.cz (D. Fišer), alto@cs.aau.dk (Á. Torralba), hoffmann@cs.uni-saarland.de (J. Hoffmann).

https://doi.org/10.1016/j.artint.2024.104174
Received 13 August 2023; Received in revised form 11 June 2024; Accepted 11 June 2024

Available online 21 June 2024

to solve such problems: heuristic explicit search and symbolic search. Heuristic explicit search guides the exploration of states using heuristic functions that estimate remaining cost. A* search [27] guarantees cost-optimality—it returns a solution whose summed-up operator cost is minimal if the heuristic is admissible. The design of admissible heuristic functions has been intensively investigated in planning [e.g., 9,28–30,41,43], and planning algorithms based on these techniques are state-of-the-art for many benchmark domains in cost-optimal planning.

In contrast to heuristic explicit search, symbolic search replaces the exploration of individual states with that of state sets, compactly represented using binary decision diagrams (BDDs) [6]. The primary operations needed for search can be implemented at the level of BDDs, in time polynomial in the size of the BDDs. This greatly improves exhaustive (blind) search, as it allows to represent and manipulate large state-space fractions efficiently [7]. Thus, symbolic search is very effective whenever large portions of the state space need to be traversed [52,53]. In cost-optimal planning, algorithms of this kind [12,14,15,57] slightly lag behind heuristic explicit search in terms of overall performance across benchmark domains, but are highly complementary and beat heuristic explicit search in a range of benchmark domains. In short, both approaches together constitute the state-of-the-art in cost-optimal planning.

In principle, the two approaches are orthogonal enhancements of the same vanilla search algorithm—state-space search—and so a natural idea is to combine the two. Indeed that idea has been presented decades ago in the BDDA* algorithm [10,18], and has received substantial attention ever since, either using heuristics to enhance symbolic search [10,16,17,26,32,33], using symbolic search to compute informative heuristics [10,11,25,60], or both [34,55,59].

Yet, this combination has not been an unqualified success. For a heuristic function h to be usable in heuristic symbolic search, *(i) h must be applicable to sets of states rather than individual ones*, as evaluating the heuristic on each state individually would defeat the purpose of symbolic search. Furthermore, as heuristic symbolic search requires to distinguish states with different heuristic values, *(ii) the partitioning of states into BDD-represented sets is different when using h, which may be detrimental for BDD size*. Condition (i) has been achieved for some strong heuristics in planning, in particular for pattern databases (PDBs) [34,60]. But it remains elusive for many other competitive heuristic functions. Regarding (ii), it has been shown that even extremely informed heuristics can exponentially deteriorate search performance [51], increasing BDD size to the extent of massively outweighing the reduction in search space size.

Due to all this, symbolic bi-directional blind search, without heuristics, is at this time considered the dominant symbolic search approach, and the use of heuristic search in this context has lost traction.

Here we challenge this trend by showing that potential heuristics [40], denoted in what follows by h^{P}, yield fresh synergy between heuristic and symbolic search. We focus on the simplest kind of potential heuristics (i.e. those of dimension one), which assign a fixed potential value $\mathrm{P}(f)$ to each fact f (i.e., each state-variable/value pair) in a given planning task, and obtain the heuristic value $h^{\mathrm{P}}(s)$ of a state s as the sum $h^{\mathrm{P}}(s) = \sum_{f \in s} \mathrm{P}(f)$ of potential values of the facts f true in s. It is ensured via linear program (LP) constraints over the fact potentials that h^{P} is an admissible and consistent heuristic function.

This family of heuristic functions does not per se satisfy condition (i). Here we show how to reformulate them in a way that addresses this problem. Our key observation is that we can express potential heuristics through a fixed *operator potential* value $\mathbb{Q}(o)$ for each of the task's operators o instead, capturing the change of heuristic value $h^{\mathrm{P}}(s') - h^{\mathrm{P}}(s)$ for any state transition $s \to s'$ induced by o. We show that, under a mild assumption on the planning task structure (discussed below), $h^{\mathrm{P}}(s)$ for a state s reached via an operator sequence $\langle o_1, \dots, o_k \rangle$ is equal to the value of h^{P} in the initial state plus the sum $\mathbb{Q}(o_1) + \cdots + \mathbb{Q}(o_k)$ of operator potentials. This heuristic function satisfies (i) in the sense that we can express the heuristic value change as part of the BDD transition relation (TR) in symbolic search steps. Specifically, this reformulated potential heuristic fits into the symbolic heuristic search algorithm GHSETA* [33], which partitions TRs by both their costs and the change of heuristic values they induce.

The assumption required for the above is that every state variable V affected by the effect of an operator o is constrained by o's precondition. This is true in many standard planning benchmarks, but is of course not true in general. For input tasks that do not satisfy this assumption, our reformulation can also be applied and still yields admissible heuristics, but these are path-dependent and inconsistent, necessitating node re-opening in search. Therefore, they do not tend to work well in practice [21,22]. For this reason and because this setting unnecessarily complicates the theory, we discuss them only in Appendix A. We present two better remedies. First, disambiguations [2,20] allow to weaken the assumption, and also yield much stronger potential heuristics. Second, one can use task transformations to transition normal form [38], where necessary, to achieve the assumption.

Another technical difficulty is that the operator potentials are real (floating-point) numbers, which can lead to rounding and precision issues. Naïvely rounding these values may lead to inconsistent heuristics. We show that this can instead be dealt with by extending the potential-heuristic LP to a mixed-integer linear program (MIP) that forces the operator potentials to be integers.

Putting the above pieces together, we obtain a new heuristic function for *forward* symbolic heuristic search: the forward search direction is enforced as computing $h^{\mathrm{P}}(s)$ requires to know the operator sequence $\langle o_1, \dots, o_n \rangle$ leading to s. This is at odds with backward search and bi-directional search, which are traditional key strengths of symbolic search. However, as it turns out, our approach applies to such searches as well, through a different reformulation where $\langle o_{k+1}, \dots, o_n \rangle$ are the operators on the path from the search state to the goal node, and $h^{\mathrm{P}}(s)$ equals the value of h^{P} in the initial state plus the sum of $\mathbb{Q}(o_i)$ over o_{k+1} to o_n, i.e., it turns out summing operator-potentials over sequences of operators works both in forward and backward direction.

This equality for the backward direction requires not only the mild assumption discussed above, but additionally requires the strong assumption that there is a single unique goal state. One can, again, apply our approach anyway to obtain path-dependent and inconsistent heuristics, but this does not always pay off in practice. What turns out to be effective instead is to partition the goal states over their heuristic values at the beginning of symbolic backward search. The operator-potentials then work analogously to

forward search. This in turn extends to symbolic *bi-directional* search where we can choose any combination of operator-potential or blind heuristics for each search direction.[1]

We run exhaustive experiments on IPC benchmarks, evaluating several different instantiations of potential heuristics in forward, backward, and bi-directional symbolic search, and comparing these configurations to the state-of-the-art in cost-optimal planning. Our operator-potential heuristics turn out to be highly beneficial. They hardly ever suffer from the risk (ii) of possibly increased BDD sizes. The key observations are:

- Our combination of symbolic search with potential heuristics vastly outperforms each of its components, showing that this combination is (much) more than the sum of its parts.
- Our best configurations soundly beat previous optimal symbolic planning algorithms, establishing a new state-of-the-art for this method family.
- Our best configurations furthermore bring symbolic search on par with the state of the art in optimal heuristic explicit search planning in overall performance, while maintaining the high level of complementarity. Thus we improve the state of the art in cost-optimal planning overall.

This paper is a combination and extension of two of our previous publications [21,22]. In [21], we introduced operator-potential heuristics and we showed how to efficiently combine them with the forward symbolic search. In [22], we addressed the application of operator-potential heuristics in the backward and bi-directional symbolic search resulting in a path-dependent inconsistent but admissible variant of operator-potential heuristics for the backward direction. In this paper, we unify the formulations of operator-potential heuristics from those two previous publications, and we present a coherent description of operator-potential heuristics and their integration to forward, backward, and bi-directional symbolic search. Moreover, we extend these findings by showing how to turn path-dependent (inconsistent) variant of operator-potential heuristics for the backward search into heuristics that are consistent, which leads to a significant improvement of the backward symbolic search. Lastly, we present a comprehensive and detailed experimental analysis of virtually all aspects of operator-potential heuristics and their integration into symbolic search.

The paper is organized as follows. We next give the necessary background on planning framework and notations, potential heuristics, and symbolic search (Section 2). We then introduce our reformulated operator-potential heuristics, analyzing possible designs for forward and backward search (Section 3). We show that these heuristics can easily be used in symbolic search (Section 4). We give a detailed empirical evaluation (Section 5) before concluding the paper (Section 6).

For ease of reading, we limit our analysis in Section 3 to the case where all effect variables are constrained by the precondition; Appendix A discusses operator-potential heuristics not making that assumption.

2. Background

We consider the finite domain representation (FDR) of planning tasks [5]. An **FDR planning task** Π is specified by a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$. $\mathcal{V}$ is a finite set of **variables**, each variable $V \in \mathcal{V}$ has a finite **domain** $\mathrm{dom}(V)$. A **fact** $\langle V, v \rangle$ is a pair of a variable $V \in \mathcal{V}$ and one of its values $v \in \mathrm{dom}(V)$. The set of all facts is denoted by $\mathcal{F} = \{\langle V, v \rangle \mid V \in \mathcal{V}, v \in \mathrm{dom}(V)\}$, and the set of facts of variable V is denoted by $\mathcal{F}_V = \{\langle V, v \rangle \mid v \in \mathrm{dom}(V)\}$. A **partial state** p is a variable assignment over some variables $\mathrm{vars}(p) \subseteq \mathcal{V}$. We write $p[V]$ for the value assigned to the variable $V \in \mathrm{vars}(p)$ in the partial state p. We also identify p with the set of facts contained in p, i.e., $p = \{\langle V, p[V] \rangle \mid V \in \mathrm{vars}(p)\}$. A partial state s is a **state** if $\mathrm{vars}(s) = \mathcal{V}$. I is an **initial state**. G is a partial state called **goal**, and a state s is a **goal state** iff $G \subseteq s$. $\mathcal{S}$ denotes the set of all states. Let p, t be partial states. We say that t **extends** p if $p \subseteq t$.

$\mathcal{O}$ is a finite set of **operators**, each operator $o \in \mathcal{O}$ has a **precondition** $\mathrm{pre}(o)$, **prevail condition** $\mathrm{prv}(o)$, and **effect** $\mathrm{eff}(o)$, which are partial states over $\mathcal{V}$, and a cost $\mathrm{cost}(o) \in \mathbb{R}_0^+$. For every operator $o \in \mathcal{O}$ it holds that $\mathrm{vars}(\mathrm{pre}(o)) \subseteq \mathrm{vars}(\mathrm{eff}(o))$, and $\mathrm{vars}(\mathrm{pre}(o)) \cap \mathrm{vars}(\mathrm{prv}(o)) = \emptyset$, and $\mathrm{vars}(\mathrm{prv}(o)) \cap \mathrm{vars}(\mathrm{eff}(o)) = \emptyset$, i.e., preconditions are defined only over affected variables, preconditions and prevail conditions are defined over a different set of variables, and prevail conditions cannot be defined over any affected variable. We also assume that $\mathrm{pre}(o)[V] \neq \mathrm{eff}(o)[V]$ for every $V \in \mathrm{vars}(\mathrm{pre}(o)) \cap \mathrm{vars}(\mathrm{eff}(o))$.

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 $o[\![s]\!]$ such that $o[\![s]\!][V] = \mathrm{eff}(o)[V]$ for every $V \in \mathrm{vars}(\mathrm{eff}(o))$, and $o[\![s]\!][V] = s[V]$ for every $V \in \mathcal{V} \setminus \mathrm{vars}(\mathrm{eff}(o))$.

Given non-negative integers $k, n \in \mathbb{N}_0$, $[k, n]$ denotes the set $\{k, \dots, n\}$ for $k \leq n$, and $[k, n]$ is defined as an empty set for $k > n$. Moreover, $[n]$ denotes a shorthand for $[1, n]$. A sum over an empty set is considered to be zero.

A sequence of operators $\pi = \langle o_1, \dots, o_n \rangle$ is applicable in a state s_0 if there are states $s_1, \dots, s_n$ 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 $\mathrm{cost}(\pi) = \sum_{i \in [n]} \mathrm{cost}(o_i)$ denotes the cost of this sequence of operators. We also consider an empty sequence of operators π which is applicable in every state s and $\pi[\![s]\!] = s$.

A sequence of operators $\pi = \langle o_1, \dots, o_n \rangle$ is called an ***s*-*t*-path** if there exist states s and t such that π is applicable in s and $\pi[\![s]\!] = t$. A sequence of operators π is called an ***s*-plan** if it is applicable in the state s and $\pi[\![s]\!]$ is a goal state, and I-plan is called simply 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, otherwise we say it is **forward unreachable**. A state s is **backward reachable** if there exists an s-plan, otherwise we say it is **backward unreachable**. An operator o is forward (backward) reachable

[1] One may consider to perform symbolic heuristic backward search simply by reversing the planning task and applying our techniques for symbolic heuristic forward search. But this would require to enumerate individual goal states, or to transform the task to the transition normal form with a single goal state. The former is obviously ineffective, and the latter is ineffective for symbolic search as we show in the experimental evaluation in Section 5.2.

iff it is applicable in some forward (backward) reachable state. The set of all forward reachable states is denoted by $\mathcal{S}_{\text{fw}}$, the set of all I-s-paths for all $s \in \mathcal{S}_{\text{fw}}$ is denoted by $\mathcal{E}_{\text{fw}}$, the set of all backward reachable states is denoted by $\mathcal{S}_{\text{bw}}$, and the set of all s-plans for all $s \in \mathcal{S}_{\text{bw}}$ is denoted by $\mathcal{E}_{\text{bw}}$. A state $s \in \mathcal{S}_{\text{fw}} \setminus \mathcal{S}_{\text{bw}}$ that is forward reachable but not backward reachable is called **forward dead-end** (i.e., forward dead-ends s are states that are reachable from the initial state, but there does not exist any s-plan), and a state $s \in \mathcal{S}_{\text{bw}} \setminus \mathcal{S}_{\text{fw}}$ that is backward reachable but not forward reachable is called **backward dead-end** (i.e., backward dead-ends s are states for which there exist an s-plan, but they are not reachable from the initial state).

A **forward heuristic** $h_{\text{fw}} : \mathcal{S}_{\text{fw}} \mapsto \mathbb{R} \cup \{\infty\}$ estimates the cost of optimal s-plans for all forward reachable states $s \in \mathcal{S}_{\text{fw}}$. The **optimal forward heuristic** $h^\star_{\text{fw}}(s)$ maps each forward reachable state s to the cost of the optimal s-plan or to ∞ if s is a forward dead-end state. A forward heuristic h_{fw} is called

1. **forward admissible** if $h_{\text{fw}}(s) \leq h^\star_{\text{fw}}(s)$ for every forward reachable state $s \in \mathcal{S}_{\text{fw}}$;
2. **goal-aware** if $h_{\text{fw}}(s) \leq 0$ for every forward reachable goal state s; and
3. **forward consistent** if $h_{\text{fw}}(s) \leq h(o[\![s]\!]) + \text{cost}(o)$ for all forward reachable states $s \in \mathcal{S}_{\text{fw}}$ and operators $o \in \mathcal{O}$ applicable in s.

A **backward heuristic** $h_{\text{bw}} : \mathcal{S}_{\text{bw}} \mapsto \mathbb{R} \cup \{\infty\}$ estimates the cost of optimal I-s-paths. The optimal backward heuristic $h^\star_{\text{bw}}(s)$ maps each backward reachable state s to the cost of the optimal I-s-path or to ∞ if s is a backward dead-end. A backward heuristic h_{bw} is called

1. **backward admissible** if $h_{\text{bw}}(s) \leq h^\star_{\text{bw}}(s)$ for every backward reachable state $s \in \mathcal{S}_{\text{bw}}$;
2. **init-aware** if $h_{\text{bw}}(I) \leq 0$;
3. **backward consistent** if $h_{\text{bw}}(o[\![s]\!]) \leq h_{\text{bw}}(s) + \text{cost}(o)$ for all backward reachable states $s \in \mathcal{S}_{\text{bw}}$ and operators $o \in \mathcal{O}$ such that o is applicable in s and $o[\![s]\!]$ is backward reachable.

Note that we allow negative heuristic values as is usual in works on potential heuristics, because it allows to find more informed potential heuristics [e.g., 40], and we can treat negative estimates as zeros during the search. Admissibility and consistency is usually defined for all states whereas here we define them for forward and backward reachable states only. Clearly, if a forward (backward) heuristic is goal-aware (init-aware) and forward (backward) consistent, then it is also forward (backward) admissible. Sometimes we omit the adjective forward or backward when it is clear from the context. In particular, admissibility and consistency of a forward heuristic will always mean forward admissibility and forward consistency, respectively, and admissibility and consistency of a backward heuristic will always mean backward admissibility and backward consistency, respectively.

We also consider heuristic functions over all states, $h : \mathcal{S} \mapsto \mathbb{R} \cup \{\infty\}$. Nevertheless, admissibility and consistency is used only for forward and backward heuristics, goal-awareness is used only for forward heuristics, and init-awareness only for backward heuristics.

In the context of heuristic search, h-value of a state node s refers to the heuristic value of s, g-value to the cost of the sequence of operators leading to s, and f-value is the sum of g-value and the maximum of h-value and zero (since we allow negative h-values).

We define heuristics as **state-dependent** meaning they are functions mapping states to numbers. We also deal with **path-dependent** heuristics that map sequences of operators to numbers, i.e., a path-dependent heuristic can return different numerical values for the same state depending on the sequence of operators that leads to it. The exact definition of path-dependent heuristics is provided in Appendix A as we deal with them formally there.

A set of facts $M \subseteq \mathcal{F}$ is a **mutex** if $M \not\subseteq s$ for every forward reachable state $s \in \mathcal{S}_{\text{fw}}$. We will leverage prior work on so-called disambiguation [2,20]. Given a variable $V \in \mathcal{V}$ and a partial state p, a set of facts $F \subseteq \mathcal{F}_V$ is called a **disambiguation of** V **for** p if for every forward reachable state $s \in \mathcal{S}_{\text{fw}}$ such that $p \subseteq s$ it holds that $F \cap s \neq \emptyset$ (i.e., $\langle V, s[V]\rangle \in F$).

Disambiguation allows us to infer which facts cannot be part of any forward reachable state extending a given partial state. For example, suppose we have three variables V_a, V_b and V_c each having two facts: $a_1, a_2 \in \mathcal{F}_{V_a}$, $b_1, b_2 \in \mathcal{F}_{V_b}$ and $c_1, c_2 \in \mathcal{F}_{V_c}$. Moreover, suppose there is no forward reachable state containing a_1 and b_1 at the same time, or b_2 and c_2 at the same time, i.e., $\{a_1, b_1\}$ and $\{b_2, c_2\}$ are mutexes. Now, given a partial state $p = \{b_1, c_1\}$, we can infer from the aforementioned mutexes that there is no forward reachable state extending p containing a_1 because every such state already contains b_1. Therefore, the set $\{a_2\}$ is a disambiguation of V_a for p. If we consider the variable V_b that is already defined in p, then we get that the set $\{b_1\}$ is a disambiguation of V_b for p, because we can safely say that any forward reachable state extending p must contain b_1. As another example, consider a partial state $p' = \{a_1, c_2\}$. In this case, we have that the empty set $\emptyset$ is a disambiguation of V_b for p', because we can infer from the mutexes that neither b_1 or b_2 can be part of any forward reachable state extending p'. Therefore, we can conclude that p' itself is a mutex as there is no forward reachable state (i.e., variable assignment over all variables) containing p'. In other words, disambiguation tells us which facts can potentially appear in forward reachable states extending a given partial state. Note that disambiguations are allowed to overapproximate these sets which is necessary because we are usually not able to find a complete set of mutexes—there can be exponentially many of them, and it is as hard as planning to prove that a given set of facts is mutex [23].

Clearly, every $\mathcal{F}_V$ is a disambiguation of V for all possible partial states, and if $\langle V, v\rangle \in p$ then $\{\langle V, v\rangle\}$ is a disambiguation of V for p. Moreover, if the disambiguation of V for p is an empty set (for any V), then all states extending p are unreachable. Therefore, we can use empty disambiguations to determine unsolvability of planning tasks (if G extends p), or to prune unreachable operators (if a precondition or prevail condition of the operator extends p). So, from now on we will consider only non-empty disambiguations, and we will assume that, for every partial state p and a variable $V \in \text{vars}(p)$, the disambiguation of V for p is exactly $\{\langle V, p[V]\rangle\}$.

Fišer et al. [20] showed how to use mutexes to find disambiguations, so here we will assume we already have disambiguations inferred. Given an operator $o \in \mathcal{O}$, $D_o(V)$ denotes a disambiguation of V for $\text{pre}(o) \cup \text{prv}(o)$, and $D_G(V)$ denotes a disambiguation

of V for the goal G. Note that as per our assumption above, we have that $D_o(V) = \{\langle V, v\rangle\}$ for every $\langle V, v\rangle \in \mathrm{pre}(o) \cup \mathrm{prv}(o)$, and $D_G(V) = \{\langle V, v\rangle\}$ for every $\langle V, v\rangle \in G$.

A planning task Π is in Transition Normal Form (TNF) if (i) $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every $o \in \mathcal{O}$ and (ii) the goal is a fully defined state. Here, we are interested only in the first condition, so we say that the planning task Π is **normalized** if $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every $o \in \mathcal{O}$. From now on, we assume the given planning task is normalized. This simplifies the presentation and proofs, but we discuss the general case in Appendix A.

Every planning task can be normalized in polynomial time by introducing new auxiliary zero-cost operators, which grow the representation only polynomially [38], and it can be further improved with disambiguations [20]. Unfortunately, this transformation turns out to be detrimental to symbolic search as we show in Section 5.2.

However, we can also use a more straightforward "multiplication" method that, for every operator $o \in \mathcal{O}$ and every of its affected variable not appearing in its precondition $V \in \mathrm{eff}(o) \setminus \mathrm{pre}(o)$, enumerates all possible values of V and creates the corresponding operators. This method can be improved with disambiguations as we do not need to enumerate all values of V, but we can consider only the disambiguation $D_o(V)$. It turns out that, despite its worst-case exponential increase in task size, it very rarely happens in our benchmarks that a task cannot be transformed with this method, and it has a good synergy with the symbolic search.

2.1. Background on potential heuristics

Potential heuristics [39,40] are defined as weighted sums over a set of simple state features that correspond to 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 sometimes called atomic potential heuristics), where all features are single facts. It assigns a numerical value to each fact, and the heuristic value for a state s is then simply a sum of the potentials of all facts in s.

Definition 1. Let Π denote a planning task with facts $\mathcal{F}$. A **potential function** is a function $\mathtt{P} : \mathcal{F} \mapsto \mathbb{R}$. A **potential heuristic** for $\mathtt{P}$ maps each state $s \in \mathcal{S}$ to the sum of potentials of facts in s, i.e.,

$$h^{\mathtt{P}}(s) = \sum_{f \in s} \mathtt{P}(f). \tag{1}$$

Moreover, we use $h^{\mathtt{P}}_{\mathrm{fw}}$ to denote $h^{\mathtt{P}}$ restricted to forward reachable states, i.e., $h^{\mathtt{P}}_{\mathrm{fw}}(s) = h^{\mathtt{P}}(s)$ for every forward reachable state $s \in \mathcal{S}_{\mathrm{fw}}$.

Now we can state sufficient conditions for the potential heuristic to be forward consistent, goal-aware, and forward admissible, which we will need later on. We use the formulation using disambiguation previously introduced by Fišer et al. [20] and adapted to our notation and the assumption that we have a normalized planning task. In contrast to the prior formulation [20, Theorem 7], we simplify the condition ensuring forward consistency (Equation (3) below), because we assume we have a normalized planning task where $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every $o \in \mathcal{O}$, i.e., for every affected variable $V \in \mathrm{vars}(\mathrm{eff}(o))$ we know exactly what is the value of V in the precondition of o (and thus also in the state where o is applicable).

Theorem 2. *Let $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G\rangle$ denote a normalized planning task with facts $\mathcal{F}$, and let $\mathtt{P}$ denote a potential function. If*

$$\sum_{V \in \mathcal{V}} \max_{f \in D_G(V)} \mathtt{P}(f) \leq 0 \tag{2}$$

and for every operator $o \in \mathcal{O}$ it holds that

$$\sum_{f \in \mathrm{pre}(o)} \mathtt{P}(f) - \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) \leq \mathrm{cost}(o), \tag{3}$$

then $h^{\mathtt{P}}_{\mathrm{fw}}$ is goal-aware, forward consistent, and forward admissible.

Equation (2) ensures goal-awareness, and Equation (3) ensures forward consistency. Note that Equation (2) uses the disambiguation $D_G(V)$ because we do not need to consider all values of every variable not appearing in the goal G, but just those that can be part of a forward reachable goal state. In practice, we can obtain potentials as a solution to a linear program (LP) with constraints corresponding to conditions from Theorem 2 [40] as follows.

1. For each $f \in \mathcal{F}$, we create a (real-valued) variable $\mathtt{P}(f)$.
2. To ensure goal-awareness, we use the constraint Equation (2). The maximization in Equation (2) can be transformed into a set of linear inequality constraints in a standard way: For every variable $V \in \mathcal{V}$, we create an auxiliary real-valued variable X_V, then for every $V \in \mathcal{V}$ and $f \in D_G(V)$, we add the constraint $\mathtt{P}(f) \leq X_V$, and finally we replace Equation (2) with the constraint $\sum_{V \in \mathcal{V}} X_V \leq 0$.
3. To ensure consistency, we add the constraint Equation (3) for every operator $o \in \mathcal{O}$.

Any solution of such LP for any objective function results in a goal-aware and forward consistent potential function. Since we can choose any objective function, we can look for potential heuristics maximizing the heuristic estimate for the initial state [40], we can maximize the average heuristic estimates for all (syntactic) states $\mathcal{S}$ [44], use mutexes to disregard some states that are not reachable [20], or we can even combine some of the above. For example, we can construct a LP so that we obtain a potential heuristic that maximizes $h^{\mathrm{P}}_{\mathrm{fw}}(I)$ while maximizing the average estimate over all states [20].

2.2. Background on symbolic search

While explicit state-space search algorithms operate on individual states, symbolic search [37] works on sets of states compactly represented as Binary Decision Diagrams (BDDs) [6]. BDDs are an efficient data-structure to represent Boolean functions $\{0,1\}^n \mapsto \{0,1\}$ in the form of a directed acyclic graph. 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 states that do not belong to S. Note that this assumes a binary encoding of states. We use the standard representation and variable ordering used in previous works on symbolic search for classical planning [34,57]. The size of a BDD B, denoted as $|B|$, refers to the number of nodes in B. The advantage of using this representation comes from the fact that BDDs can be exponentially smaller than the number of states they represent.

Once we have sets of states represented as BDDs, we can use operations on BDDs to operate with sets of states without enumerating them one by one. Operations like the union ($\cup$), intersection ($\cap$), and complement of sets of states correspond to the disjunction ($\vee$), conjunction ($\wedge$), and negation ($\neg$) of their characteristic functions, respectively. For example, if we have two BDDs B_1 and B_2, representing sets of states S_1 and S_2, the operation $B_1 \wedge B_2$ results in a BDD which represents $S_1 \cap S_2$. These operations take only polynomial time in the size of the input BDDs $O(|B_1||B_2|)$, which enables efficient manipulation of large sets of states.

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 [33,57,58]. Moreover, mutexes can be used for a more accurate approximation of reachable states [54,57].

Having a representation for sets of states as well as sets of operators, one can efficiently perform forward search by iteratively applying the *image* operation starting with the BDD representing the initial state. 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. 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 the 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 [13].

For the search in the backward direction, one can start with the BDD representing all goal states and use the operation *pre-image* instead of *image*, i.e., $\text{pre-image}(S, T_c)$ computes the set of all predecessor states S' from which a state in S can be reached by applying an operator represented by T_c. Torralba et al. [57] provide a comprehensive description of how to efficiently implement image and pre-image operations.

The most prominent implementation of symbolic heuristic search in the context of automated planning is BDDA* [18] which is a variant of A* [27] using BDDs to represent sets of states. Like A*, BDDA* expands states by ascending order of their f-value. To take advantage of the symbolic representation, BDDA* represents all states with the same g and h value in a single BDD $S_{g,h}$ (disjunctive partitioning of $S_{g,h}$ can also be used). Given a set of states $S_{g,h}$ and a TR T_c, the g-value of the resulting set of successor states $\mathrm{image}(S_{g,h}, T_c)$ is simply $g + c$. However, these successor states have to be split according to their h-value. This can usually be performed efficiently with, e.g., symbolic pattern databases [34], by partitioning all states into BDDs S_h, where each S_h represents the set of all states with the heuristic value h. Then a conjunction of the successor states and a set S_h will give us the sub-set of successor states with heuristic value h. To fully partition a set of states according to their heuristic value, we then need to compute such a conjunction for every partition S_h.

GHSETA* [33] encodes the heuristic function as part of the transition relation, creating multiple TRs depending on the impact of the operators on heuristic value. That is, we need a function $\delta_h : \mathcal{O} \mapsto \mathbb{R}$ mapping operators to numbers so that if the heuristic value for the state s is $h(s)$ and the operator $o \in \mathcal{O}$ is applicable in s, then $h(o[\![s]\!]) = h(s) + \delta_h(o)$ is the heuristic value for the successor state $o[\![s]\!]$. Then we can partition operators into TRs not only by their costs but also by the change of the heuristic value $\delta_h(o)$ they induce, i.e., instead of having a TR T_c for every operator with the cost c, we have a TR $T_{c,q}$ for every operator cost c and every possible value $q = \delta_h(o)$. With this approach, computing g and h-values of successor states is much more straightforward than in the previous case: $\mathrm{image}(S_{g,h}, T_{c,q})$ 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 $h + q$. This is a very efficient way of evaluating the heuristics within symbolic search. However, up to now, all heuristics known to be suitable for this representation were either non-informative, inadmissible, or domain dependent. We show, in the next two sections, that potential heuristics can be adapted to this schema to smoothly integrate them into the GHSETA* algorithm.

Algorithm 1 shows the pseudo-code of the GHSETA* algorithm in the forward direction. It takes a planning task, a heuristic estimate h_I for the initial state, and a function δ_h inducing, together with h_I, a consistent admissible forward heuristic, i.e., we assume that for every sequence of operators $\pi = \langle o_1, \ldots, o_n\rangle$ applicable in I it holds that $h_I + \sum_{i \in [n]} \delta_h(o_i)$ is a forward consistent and forward admissible heuristic estimate for the state $\pi[\![I]\!]$.

Algorithm 1: GHSETA* in the forward direction with a consistent heuristic.

Input: A planning task Π, a heuristic estimate $h_I \geq 0$ for the initial state, and a function $\delta_h : \mathcal{O} \mapsto \mathbb{R}$ so that h_I and δ_h induce a consistent and admissible heuristic.
Output: An optimal plan or "unsolvable".

```
1  for each ⟨c, q⟩ ∈ {⟨cost(o), δ_h(o)⟩ | o ∈ O} do
2  |  Construct T_{c,q} from {o ∈ O | cost(o) = c, δ_h(o) = q} ;
3  S_{0,h_I} ← BDD representing the set {I} ;
4  open ← {⟨h_I, S_{0,h_I}⟩};
5  closed ← ∅ ;
6  while open ≠ ∅ do
7  |  ⟨f, S_{g,h}⟩ ← PopMin(open);
8  |  S_{g,h} ← S_{g,h} ∧ ¬closed ;
9  |  if S_{g,h} contains a goal state then
10 |  |  return ExtractPlan(S_{g,h}) ;
11 |  closed ← closed ∨ S_{g,h} ;
12 |  for each T_{c,q} do
13 |  |  S_{g+c,h+q} ← image(S_{g,h}, T_{c,q}) ∧ ¬closed ;
14 |  |  if S_{g+c,h+q} ≠ ∅ then
15 |  |  |  f ← g + c + max(0, h + q) ;
16 |  |  |  open ← InsertOrUpdate(open, f, S_{g+c,h+q}) ;
17 return "unsolvable";

18 function InsertOrUpdate(O, f, S_{g,h})
19 |  if there exists ⟨f, S'_{g,h}⟩ ∈ O then
20 |  |  return (O \ ⟨f, S'_{g,h}⟩) ∪ {⟨f, S'_{g,h} ∨ S_{g,h}⟩};
21 |  else
22 |  |  return O ∪ {⟨f, S_{g,h}⟩};
```

Lines 1 and 2 describe the partitioning of operators into TRs based on their cost and the change of heuristic value they induce via the function δ_h. The rest is a standard A* algorithm without re-opening states (because we assume a consistent heuristic) adapted for searching over sets of states and computing heuristic values by summing over sequences of operators rather than calling a heuristic function for every expanded state. The main distinctions to the state-space A* are the following:

1. As in the standard A*, we maintain the set of closed states (lines 5 and 11). However, since we operate on sets of states instead of individual states, we represent the set of closed states as a BDD (possibly with disjunctive partitioning), and we skip closed states stored in the set `closed` by removing `closed` from all expanded and generated sets of states (lines 8 and 13).
2. GHSETA* also maintains an open list as a priority queue ordering states by the increasing f-values, but all states with the same g and h-values are merged into one BDD (line 16 and the function `InsertOrUpdate()`). So, in each cycle, a set of states with the lowest f-value are processed at once using the BDD $S_{g,h}$ with minimal g-value among those with minimal $f = g + \max(h, 0)$ value.
3. Given a set of states $S_{g,h}$ (with the g-value g and h-value h) and a TR $T_{c,q}$ (with the cost c and inducing the change of h-value by q), we can easily compute the g and h-value of the successor states image$(S_{g,h}, T_{c,q})$ as $g + c$ and $h + q$, respectively (line 13).
4. Instead of terminating when a goal state is removed from the queue, we terminate when we remove a set of states containing a goal state. The plan extraction in GHSETA* (line 10) is a little bit more complicated than in state-space A*, because a simple backchaining from a goal state is not possible here. Nevertheless, it is still polynomial in the length of the plan—a detailed description is provided by Torralba et al. [57].

Finally, we adapt the GHSETA* algorithm to support negative h-values. Instead of considering the f-value of a state to be $g + h$, we use instead $f = g + \max(h, 0)$. Therefore, the f-value of the successor states is $g + c + \max(0, h + q)$ (in line 15). This is not only an optimization (i.e., avoiding the expansion of any bucket where $g > h^\star_{\text{fw}}(I)$ even if $g + h < h^\star_{\text{fw}}(I)$). In fact, this is also needed for correctness of the stopping condition, as otherwise goal states with negative heuristic value could be expanded even if they do not correspond to an optimal plan. The common solution of simply changing the heuristic function to $\max(h, 0)$ is not possible as that cannot always be expressed as a δ_h function. However, by keeping the original (negative) h value for the BDD representation, and making the heuristic non-negative only when computing the f-value, we get the best of both worlds: an efficient BDD representation without unnecessarily expanding any set of states with negative heuristic value.

Note that Jensen et al. [33] also introduce the FSETA* algorithm where the change of heuristic values is compiled directly into operators' costs. This is similar to encoding the heuristic as a task transformation, i.e., by changing the cost of each operator to be $\text{cost}(o) + \delta_h(o)$. It is well-known that running Dijkstra on the reformulated task is equivalent to running A* on the original task [36]. However, it is not entirely clear how to apply these approaches in the presence of heuristics that can take negative heuristic values. We leave this question to future research.

If no heuristic is used (i.e., $h = 0$ for all states), performing *backward* search is straightforward. One can simply run Algorithm 1, starting with the BDD representing the set of all goal states instead of the initial state (line 3). Then, at each step, it uses the pre-image operation instead of image (line 13), and it terminates when a BDD containing the initial state is removed from the priority queue (line 9). In Sections 3 and 4, we explain how to extend this for performing backward search with any (backward) consistent and admissible heuristic.

The *bi-directional* search combines the forward and backward search by keeping separate open and closed lists for each direction and then alternating between the forward and backward steps. In each iteration of the algorithm, it is decided whether to expand a set of states from the forward or the backward open list. A common criterion is to select the search direction whose next step is estimated to be easiest (e.g., by selecting the set of states whose BDD representation is smallest). In our implementation, we use the criteria used by Torralba et al. [57], which besides the BDD size, also considers the time spent in previous iterations to estimate which direction will take less time in completing the next step.

The bi-directional search stops when both directions meet and we are able to prove that the plan combined from both directions is an optimal plan. That is, instead of checking whether the current set of states selected for expansion contains a goal state (line 9), we check whether the intersection with the closed list from the opposite direction is non-empty. If the intersection is not empty, then any state in such intersection is part of a plan. The algorithm keeps track of the best plan π found so far, and terminates as soon as no better plan can be found, i.e., whenever $\mathrm{cost}(\pi) \leq \min_{s \in \mathtt{open}_f} f(s)$ or $\mathrm{cost}(\pi) \leq \min_{s \in \mathtt{open}_b} f(s)$ or $\mathrm{cost}(\pi) \leq \min_{s \in \mathtt{open}_f} g(s) + \min_{s \in \mathtt{open}_b} g(s)$, where $\mathtt{open}_f$ and $\mathtt{open}_b$ are the open lists of the forward and backward search, respectively, and $f(s)$ and $g(s)$ denote f and g-values of a state s, respectively.[2] This guarantees that the bi-directional search terminates with an optimal plan, as long as an admissible heuristic is used in both directions, even when different heuristics are used in each direction. Note also that each direction can use a different partitioning of operators into TRs.

3. Operator-potential heuristics

Potential heuristics map facts to numerical values. Here, we show that instead of mapping facts to numerical values, we can map each operator o to a numerical value, called operator-potential, corresponding to the change of the heuristic value over a transition induced by o. More precisely, we show how to transform a potential function $\mathtt{P} : \mathcal{F} \mapsto \mathbb{R}$ to an operator-potential function $\mathtt{Q} : \mathcal{O} \mapsto \mathbb{R}$ so that for every state s and each operator o applicable in s it holds that $h^{\mathtt{P}}(o[\![s]\!]) = h^{\mathtt{P}}(s) + \mathtt{Q}(o)$. In other words, we define $\mathtt{Q}$ in such a way that $\mathtt{Q}(o)$ is exactly equal to the change of heuristic value of the corresponding potential heuristic over a transition between states induced by the operator o.

Recall that we assume $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every operator $o \in \mathcal{O}$ (the general case is discussed in Appendix A). As pointed out by [45] in the context of proving the limitations of one-dimensional potential heuristics, this means that we know exactly how each operator o changes the state s on which it is applied, i.e., for every fact $\langle V, v \rangle \in \mathrm{eff}(o)$ we know exactly what is the value $s[V]$ because $\langle V, s[V] \rangle \in \mathrm{pre}(o)$. (Note that the same is not true for higher-dimensional potential heuristics, so operator-potential functions are defined for potential heuristics of dimension one only.)

Definition 3. Given a potential function $\mathtt{P}$, a function $\mathtt{Q} : \mathcal{O} \mapsto \mathbb{R}$ is called an **operator-potential function** for $\mathtt{P}$ if

$$\mathtt{Q}(o) = \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) - \sum_{f \in \mathrm{pre}(o)} \mathtt{P}(f) \tag{4}$$

for every operator $o \in \mathcal{O}$.

In the following proposition, we show that $\mathtt{Q}(o)$ is exactly equal to the change of the heuristic value of the potential heuristic from a state to state. Note that Proposition 4 holds for any state s, in particular, for every forward reachable as well as every backward reachable state.

Proposition 4. *Let $s \in \mathcal{S}$ denote a state, and let $o \in \mathcal{O}$ denote an operator applicable in s. Then $\sum_{f \in s} \mathtt{P}(f) + \mathtt{Q}(o) = \sum_{f \in o[\![s]\!]} \mathtt{P}(f)$.*

Proof. Let $t = s \setminus \mathrm{pre}(o)$. Since we assume $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$, it follows that $t = o[\![s]\!] \setminus \mathrm{eff}(o)$. Therefore, we have that

$$\begin{aligned}
\sum_{f \in s} \mathtt{P}(f) + \mathtt{Q}(o) &= \sum_{f \in t} \mathtt{P}(f) + \sum_{f \in \mathrm{pre}(o)} \mathtt{P}(f) + \mathtt{Q}(o) \\
&= \sum_{f \in t} \mathtt{P}(f) + \sum_{f \in \mathrm{pre}(o)} \mathtt{P}(f) + \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) - \sum_{f \in \mathrm{pre}(o)} \mathtt{P}(f) \\
&= \sum_{f \in t} \mathtt{P}(f) + \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) = \sum_{f \in o[\![s]\!]} \mathtt{P}(f). \quad \square
\end{aligned}$$

[2] Recent work on bi-directional explicit-state heuristic search [1,3,31,46,47] has derived stronger bounds when consistent heuristics are used. Transferring those to symbolic search is a promising avenue for future research.

Next, we show that the property from Proposition 4 extends over sequences of operators. That is, for any two states $s, s' \in \mathcal{S}$ and any sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ leading from s to s' (i.e., π is applicable in s and $\pi[\![s]\!] = s'$) it holds that the sum over operator potentials of operators from the sequence π, $\sum_{i \in [n]} \mathtt{Q}(o_i)$, is exactly equal to the change of heuristic value of the potential heuristic from s to s', i.e., $h^{\mathtt{P}}(s) + \sum_{i \in [n]} \mathtt{Q}(o_i) = h^{\mathtt{P}}(s')$. Note that this property holds for *any* sequence of operators π between states s and s'. In other words, for a fixed pair of states $s, s' \in \mathcal{S}$ and any two sequences of operators $\pi = \langle o_1, \ldots, o_n \rangle$ and $\pi' = \langle q_1, \ldots, q_m \rangle$ both leading from s to s', it holds that $h^{\mathtt{P}}(s) + \sum_{i \in [n]} \mathtt{Q}(o_i) = h^{\mathtt{P}}(s) + \sum_{i \in [m]} \mathtt{Q}(q_i) = h^{\mathtt{P}}(s')$. Therefore for any such π and π' the sums over operator potentials are exactly the same, i.e., $\sum_{i \in [n]} \mathtt{Q}(o_i) = \sum_{i \in [m]} \mathtt{Q}(q_i)$. Therefore, summing operator potentials over sequences of operators preserves state-dependency as long as the planning task is normalized. We will use this property later when we define state-dependent operator-potential heuristics in forward and backward direction.

Proposition 5. *Let $s \in \mathcal{S}$ denote a state, and let $\pi = \langle o_1, \ldots, o_n \rangle$ denote a sequence of operators applicable in s. Then $\sum_{f \in s} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) = \sum_{f \in \pi[\![s]\!]} \mathtt{P}(f)$.*

Proof. (By induction) The claim clearly holds for the empty sequence π. Now, assume the claim holds for some sequence of operators $\pi' = \langle o_1, \ldots, o_{k-1} \rangle$ such that π' is applicable in s and $k \leq n$, and we prove that it also holds for the sequence of operators $\pi'' = \langle o_1, \ldots, o_{k-1}, o_k \rangle$. Let $s_{k-1} = \pi'[\![s]\!]$ and $s_k = \pi''[\![s]\!]$.

From Proposition 4 we have that $\sum_{f \in s_{k-1}} \mathtt{P}(f) + \mathtt{Q}(o_k) = \sum_{f \in s_k} \mathtt{P}(f)$ because o_k is applicable in s_{k-1} and $o_k[\![s_{k-1}]\!] = s_k$, and from the assumption we have that $\sum_{f \in s_{k-1}} \mathtt{P}(f) = \sum_{f \in s} \mathtt{P}(f) + \sum_{i \in [k-1]} \mathtt{Q}(o_i)$, therefore it follows that

$$\sum_{f \in s_k} \mathtt{P}(f) = \sum_{f \in s_{k-1}} \mathtt{P}(f) + \mathtt{Q}(o_k) = \sum_{f \in s} \mathtt{P}(f) + \sum_{i \in [k-1]} \mathtt{Q}(o_i) + \mathtt{Q}(o_k) = \sum_{f \in s} \mathtt{P}(f) + \sum_{i \in [k]} \mathtt{Q}(o_i),$$

which concludes the proof. □

Now that we have shown how to define operator-potential functions and we proved their fundamental properties in relation to the corresponding potential heuristics, we move to the introduction of a new family of operator-potential heuristics in forward and backward direction. In Section 3.1, we show how to construct operator-potential *forward* heuristics that are goal-aware, forward consistent and thus also forward admissible. In Section 3.2, we focus on operator-potential *backward* heuristics. We show that the same approach used for the operator-potential forward heuristics can be used also in the backward direction. Although it leads to backward admissible estimates, it can also result in path-dependent heuristics. So, we show how to remedy this issue and obtain operator-potential *backward* heuristics that are state-dependent, init-aware, backward consistent, and backward admissible.

3.1. Forward direction

Under the assumption that $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every operator and with Proposition 5 in place, the construction of a forward consistent operator-potential heuristic $h^{\mathtt{Q}}_{\mathrm{fw}}$ is straightforward. Given a potential function $\mathtt{P}$ and its corresponding operator-potential function $\mathtt{Q}$, we start by setting the heuristic value for the initial state $h^{\mathtt{Q}}_{\mathrm{fw}}(I)$ to $h^{\mathtt{P}}_{\mathrm{fw}}(I) = \sum_{f \in I} \mathtt{P}(f)$, and then it follows from Proposition 5 that adding a sum of operator-potentials over any sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ applicable in the initial state results in the heuristic value $h^{\mathtt{P}}_{\mathrm{fw}}(\pi[\![I]\!])$, i.e., such construction exactly preserves heuristic values of the potential heuristic $h^{\mathtt{P}}_{\mathrm{fw}}$ along with its properties such as consistency, goal-awareness, and admissibility.

Definition 6. Let $\mathtt{Q}$ denote an operator-potential function for $\mathtt{P}$. An **operator-potential forward heuristic** $h^{\mathtt{Q}}_{\mathrm{fw}} : \mathcal{S}_{\mathrm{fw}} \mapsto \mathbb{R} \cup \{\infty\}$ for $\mathtt{Q}$ is defined as

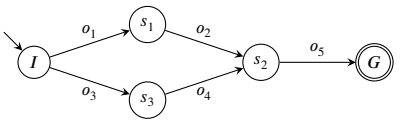
$$h^{\mathtt{Q}}_{\mathrm{fw}}(s) = \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) \tag{5}$$

for every sequence of operators $\pi = \langle o_1, \ldots, o_n \rangle$ such that $\pi[\![I]\!] = s$.

Now we need to show that $h^{\mathtt{Q}}_{\mathrm{fw}}$ is well-defined, i.e., Equation (5), indeed, expresses a function mapping forward-reachable states to numbers. In other words, we need to show that $h^{\mathtt{Q}}_{\mathrm{fw}}(s)$ is the same for every sequence of operators π leading from the initial state to s. This follows directly from Proposition 5 as it also shows that $h^{\mathtt{Q}}_{\mathrm{fw}}$ is exactly equal to $h^{\mathtt{P}}_{\mathrm{fw}}$ and therefore $h^{\mathtt{Q}}_{\mathrm{fw}}$ has exactly the same properties as $h^{\mathtt{P}}_{\mathrm{fw}}$.

Theorem 7. *$h^{\mathtt{Q}}_{\mathrm{fw}}$ is well-defined, and $h^{\mathtt{Q}}_{\mathrm{fw}}(s) = h^{\mathtt{P}}_{\mathrm{fw}}(s)$ for every forward-reachable state $s \in \mathcal{S}_{\mathrm{fw}}$, and $h^{\mathtt{Q}}_{\mathrm{fw}}$ is forward admissible (goal-aware, forward consistent) if $h^{\mathtt{P}}_{\mathrm{fw}}$ is forward admissible (goal-aware, forward consistent).*

Proof. Let $s \in \mathcal{S}_{\mathrm{fw}}$ denote a forward reachable state, and let $\pi = \langle o_1, \ldots, o_n \rangle$ denote a sequence of operators such that π is applicable in I and $\pi[\![I]\!] = s$. From Proposition 5 it follows that $\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) = \sum_{f \in s} \mathtt{P}(f)$, and from definitions of $h^{\mathtt{Q}}_{\mathrm{fw}}$ and $h^{\mathtt{P}}_{\mathrm{fw}}$ it further follows that



$o \in \mathcal{O}$	cost(o)	$\mathbb{Q}(o)$
o_1	0	1
o_2	0	0
o_3	0	0.9
o_4	0	0.1
o_5	1	−1

Fig. 1. A simple example showing path-dependency of an operator-potential heuristic after rounding operator potentials down to the nearest integers. Let I and G denote the initial and goal state, respectively, let $\text{cost}(o_i) = 0$ for all $i \in [4]$, and $\text{cost}(o_5) = 1$, and let $\mathbb{Q}(o_1) = 1$, $\mathbb{Q}(o_2) = 0$, $\mathbb{Q}(o_3) = 0.9$, $\mathbb{Q}(o_4) = 0.1$, and $\mathbb{Q}(o_5) = -1$, and let $h^{\mathbb{Q}}_{\text{fw}}(I) = 0$.

$$h^{\mathbb{Q}}_{\text{fw}}(s) = \sum_{f \in I} \mathbb{P}(f) + \sum_{i \in [n]} \mathbb{Q}(o_i) = \sum_{f \in s} \mathbb{P}(f) = h^{\mathbb{P}}_{\text{fw}}(s).$$

Therefore $h^{\mathbb{Q}}_{\text{fw}}$ is well-defined, $h^{\mathbb{Q}}_{\text{fw}}(s) = h^{\mathbb{P}}_{\text{fw}}(s)$ for every $s \in \mathcal{S}_{\text{fw}}$, and therefore if $h^{\mathbb{P}}_{\text{fw}}$ is forward admissible (goal aware, forward consistent), then so is $h^{\mathbb{Q}}_{\text{fw}}$. □

Note that $h^{\mathbb{Q}}_{\text{fw}}$ is a state-dependent heuristic even though it is computed from I-s-paths. This is because every I-s-path results in exactly the same value of $h^{\mathbb{Q}}_{\text{fw}}(s)$. Moreover, note that $h^{\mathbb{Q}}_{\text{fw}}$ can be used in an incremental way: For every forward reachable state $s \in \mathcal{S}_{\text{fw}}$ and an operator $o \in \mathcal{O}$ applicable in s, we have that $h^{\mathbb{Q}}_{\text{fw}}(o[\![s]\!]) = h^{\mathbb{Q}}_{\text{fw}}(s) + \mathbb{Q}(o)$. In other words, $h^{\mathbb{Q}}_{\text{fw}}$ can be used in search so that we assign the heuristic value $h^{\mathbb{Q}}_{\text{fw}}(I) = \sum_{f \in I} \mathbb{P}(f)$ to the initial state and then whenever we expand a state s with an operator o, we compute the heuristic value for the resulting state simply by adding $\mathbb{Q}(o)$ to the heuristic value we have previously stored for s, i.e., $h^{\mathbb{Q}}_{\text{fw}}(o[\![s]\!]) = h^{\mathbb{Q}}_{\text{fw}}(s) + \mathbb{Q}(o)$. This property of $h^{\mathbb{Q}}_{\text{fw}}$ will be particularly useful in the context of symbolic search.

As we show later, a frictionless application of operator-potentials in symbolic search requires partitioning of operators using $\mathbb{Q}(o)$ values, i.e., we need to group together operators that induce the same change of operator-potential heuristic values. Therefore, we need to compare $\mathbb{Q}(o)$ values on equality. However, $\mathbb{P}$ is typically inferred using a linear program which results in $\mathbb{Q}(o)$ values represented as floating-point numbers. This could significantly reduce efficiency of the partitioning as each partition can consist of a single operator even when $\mathbb{Q}(o)$ values differ only slightly. Moreover, the strength of symbolic search lies in its ability to aggregate states with the same heuristic and g-values into a BDD. Therefore, having floating-point heuristic values is an even larger problem.

Assuming operator costs are integers, both issues can be resolved if all $\mathbb{Q}(o)$ values and $h^{\mathbb{P}}_{\text{fw}}(I)$ are integers. It is easy to see that $h^{\mathbb{P}}_{\text{fw}}(I)$ can be safely rounded up to the nearest integer, because if costs of operators are integers, then also costs of plans must be integer-valued. It may also seem that rounding operator potentials down to the nearest integer may resolve this issue as the sums over the rounded operator potentials would result in admissible estimates. However, rounding $\mathbb{Q}(o)$ values down may result in path-dependent estimates.

Consider the planning task depicted in Fig. 1. The operator-potential heuristic $h^{\mathbb{Q}}_{\text{fw}}$ is clearly forward consistent, goal-aware, and forward admissible. Let $\hat{\mathbb{Q}}$ denote a function mapping each operator o_i to $\mathbb{Q}(o_i)$ rounded down to the nearest integer, i.e., $\hat{\mathbb{Q}}(o_1) = 1$, $\hat{\mathbb{Q}}(o_2) = \hat{\mathbb{Q}}(o_3) = \hat{\mathbb{Q}}(o_4) = 0$, and $\hat{\mathbb{Q}}(o_5) = -1$. Now, consider the state s_2. If we use $\hat{\mathbb{Q}}$ instead of $\mathbb{Q}$ to compute heuristic values using Equation (5), then taking the path $\langle o_1, o_2 \rangle$ results in $h^{\mathbb{Q}}_{\text{fw}}(I) + \hat{\mathbb{Q}}(o_1) + \hat{\mathbb{Q}}(o_2) = 1$, and the path $\langle o_3, o_4 \rangle$ results in $h^{\mathbb{Q}}_{\text{fw}}(I) + \hat{\mathbb{Q}}(o_3) + \hat{\mathbb{Q}}(o_4) = 0$, i.e., we get two different values depending on the path used to reach s_2. Moreover, note that path-dependency may also result in inconsistency: Consider the states s_1 and s_2, and operator sequence $\langle o_1 \rangle$ used to reach s_1 and $\langle o_3, o_4 \rangle$ used to reach s_2. In this case, the estimate with $\hat{\mathbb{Q}}$ for s_1 would be 1, but the estimate for s_2 would be 0 as well as the cost of o_2.

We resolve this issue by restricting the potential functions to always result in integer-valued operator-potentials. Note that this approach still allows to round $h^{\mathbb{P}}_{\text{fw}}(I)$ up to the nearest integer as it clearly preserves forward consistency, goal-awareness, and forward admissibility of the resulting heuristics.

To obtain integer operator-potentials, we propose to use the following mixed-integer linear program (MIP):

1. For every fact $f \in \mathcal{F}$, we create the real-valued variable $\mathbb{P}(f)$.
2. For every operator $o \in \mathcal{O}$, we create the integer-valued variable $\mathbb{Q}(o)$.
3. To ensure goal-awareness and forward consistency of the resulting potential function $\mathbb{P}$, we add the constraint Equation (2), and, for every $o \in \mathcal{O}$, we add the constraint Equation (3).
4. For every operator $o \in \mathcal{O}$, we add the constraint Equation (4). This ensures that $\mathbb{Q}(o)$ will be, indeed, an operator-potential as per Definition 3, and since $\mathbb{Q}(o)$ is an integer-valued variable, the resulting operator-potential will be also integer.

Clearly, any solution to such MIP results in an operator-potential function according to Equation (4) with integer $\mathbb{Q}(o)$ values and the corresponding $h^{\mathbb{Q}}_{\text{fw}}$ will be forward consistent, goal-aware, and forward admissible. So, we can use any optimization criteria that was previously proposed for the potential heuristics [20,40,44].

The disadvantage of using MIP is that it is harder to solve than LP because MIP is NP-hard in general whereas LP can be solved by a polynomial algorithm. However, this seems to be rarely a bottleneck in practice as we show in the experimental evaluation in Section 5.1.

$\mathcal{V} = \{v_1, v_2\}$, $\mathrm{dom}(v_1) = \{x, y\}$, $\mathrm{dom}(v_2) = \{A, B, C\}$

$I = \{\langle v_1, x\rangle, \langle v_2, A\rangle\}$, $G = \{\langle v_2, C\rangle\}$

$o \in \mathcal{O}$	prv(o)	pre(o)	eff(o)	cost(o)	$\mathbb{Q}(o)$
o_1	$\emptyset$	$\langle v_1, x\rangle, \langle v_2, A\rangle$	$\langle v_1, y\rangle, \langle v_2, B\rangle$	1	-1
o_2	$\emptyset$	$\langle v_2, B\rangle$	$\langle v_2, C\rangle$	1	-1
o_3	$\emptyset$	$\langle v_1, y\rangle$	$\langle v_1, x\rangle$	1	-1

f	$\mathbb{P}(f)$
$\langle v_1, x\rangle$	0
$\langle v_1, y\rangle$	1
$\langle v_2, A\rangle$	2
$\langle v_2, B\rangle$	0
$\langle v_2, C\rangle$	-1

$h^{\mathbb{P}}_{\mathrm{fw}}(I) = h^{\mathbb{Q}}_{\mathrm{fw}}(I) = 2$

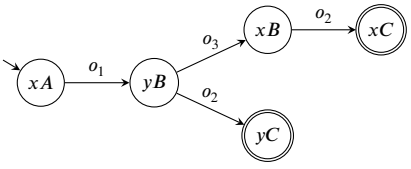


Fig. 2. Example planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G\rangle$ illustrating operator-potential backward heuristic.

3.2. Backward direction

Interestingly, under certain conditions, the very same operator-potential function and Equation (5) can be used to obtain backward admissible estimates also in the backward direction. To be more precise, given a potential function $\mathbb{P}$ such that the conditions from Theorem 2 hold and the operator-potential function $\mathbb{Q}$ for $\mathbb{P}$, for every s-plan $\pi = \langle o_1, \dots, o_n\rangle$, the estimate

$$\sum_{f \in I} \mathbb{P}(f) + \sum_{i \in [n]} \mathbb{Q}(o_i) \tag{6}$$

is a lower bound on the cost of every I-s-path. That is, Equation (6) is a backward admissible heuristic estimate for the backward search even when $\mathbb{P}$ and $\mathbb{Q}$ are computed in the same way as for the forward direction which we prove in the following Proposition 8.

Proposition 8. *Let $\mathbb{P}$ denote a potential function such that Equation (2) and Equation (3) hold, let $\mathbb{Q}$ denote an operator-potential function for $\mathbb{P}$, let $s \in \mathcal{S}_{\mathrm{fw}} \cap \mathcal{S}_{\mathrm{bw}}$ denote a state that is both forward and backward reachable, let $\pi = \langle o_1, \dots, o_n\rangle$ denote an s-plan, and let $\pi' = \langle o'_1, \dots, o'_m\rangle$ denote an I-s-path. Then $\sum_{f \in I} \mathbb{P}(f) + \sum_{i \in [n]} \mathbb{Q}(o_i) \leq \sum_{i \in [m]} \mathrm{cost}(o'_i)$.*

Proof. Note that $h^{\mathbb{Q}}_{\mathrm{fw}}(I) = \sum_{f \in I} \mathbb{P}(f)$ and that $\langle o'_1, \dots, o'_m, o_1, \dots, o_n\rangle$ is a plan. Let $g = \pi[\![s]\!]$. From Equation (2) it follows that $h^{\mathbb{P}}_{\mathrm{fw}}(g) \leq 0$, and from Theorem 7 it follows that $h^{\mathbb{Q}}_{\mathrm{fw}}(I) + \sum_{i \in [n]} \mathbb{Q}(o_i) + \sum_{i \in [m]} \mathbb{Q}(o'_i) = h^{\mathbb{P}}_{\mathrm{fw}}(g) \leq 0$. Therefore we have that $h^{\mathbb{Q}}_{\mathrm{fw}}(I) + \sum_{i \in [n]} \mathbb{Q}(o_i) \leq -\sum_{i \in [m]} \mathbb{Q}(o'_i)$. Finally, From Definition 3 and Equation (3) it follows that $-\mathbb{Q}(o'_i) \leq \mathrm{cost}(o'_i)$ for every $i \in [m]$ and therefore $-\sum_{i \in [m]} \mathbb{Q}(o'_i) \leq \sum_{i \in [m]} \mathrm{cost}(o'_i)$ which concludes the proof. □

Now, it may seem that we are ready to formulate the backward variant of operator-potential heuristics. Unfortunately, the aforementioned Equation (6) can result in different values depending on the given s-plan π, i.e., such heuristic estimates are path-dependent. Consider the planning task depicted in Fig. 2, and the state "yB" backward-reachable from both goal states "xC" and "yC". For "yC" and $\langle o_2\rangle$, Equation (6) evaluates to $h^{\mathbb{Q}}_{\mathrm{fw}}(I) + \mathbb{Q}(o_2) = 2 - 1 = 1$. And for "$xC$" and $\langle o_3, o_2\rangle$, it evaluates to $h^{\mathbb{Q}}_{\mathrm{fw}}(I) + \mathbb{Q}(o_3) + \mathbb{Q}(o_2) = 2 - 2 = 0$. Both are backward admissible estimates for the backward search as the cost of the remaining operator o_1 is 1, but the estimates are path-dependent. This behavior is caused by the fact that we allow negative heuristic estimates possibly resulting in different heuristic values of different goal states. For example, $h^{\mathbb{Q}}_{\mathrm{fw}}$-value for "$xC$" is -1 whereas $h^{\mathbb{Q}}_{\mathrm{fw}}$-value for "$yC$" is zero.

This observation suggests that we can fix this issue by incorporating heuristic values of goal states in the computation. And indeed, it turns out that if Equation (3) holds for the potential function $\mathbb{P}$, then subtracting the sum of potentials over goal state facts from Equation (6) resolves the issue. So, we define *goal-corrected operator-potential backward heuristic* accordingly and then we prove that it is well-defined (i.e., state-dependent), init-aware, backward consistent, and therefore also backward admissible.

Definition 9. Let $\mathbb{Q}$ denote an operator-potential function for $\mathbb{P}$ such that Equation (3) holds for $\mathbb{P}$. A **goal-corrected operator-potential backward heuristic** $h^{\mathbb{Q}}_{\mathrm{bw}} : \mathcal{S}_{\mathrm{bw}} \mapsto \mathbb{R} \cup \{\infty\}$ for $\mathbb{Q}$ is defined as

$$h^{\mathbb{Q}}_{\mathrm{bw}}(s) = \sum_{f \in I} \mathbb{P}(f) + \sum_{i \in [n]} \mathbb{Q}(o_i) - \sum_{f \in \pi[\![s]\!]} \mathbb{P}(f) \tag{7}$$

for every backward reachable state $s \in \mathcal{S}_{\mathrm{bw}}$ and every s-plan $\pi = \langle o_1, \dots, o_n\rangle \in \mathcal{E}_{\mathrm{bw}}$.

We start by showing in the following Lemma 10 that $h^{\mathbb{Q}}_{\mathrm{bw}}$ from Definition 9 is state-dependent, i.e., for every backward reachable state $s \in \mathcal{S}_{\mathrm{bw}}$, $h^{\mathbb{Q}}_{\mathrm{bw}}(s)$ evaluates to the same value for all s-plans. This is a consequence of Proposition 5 because it shows that for a

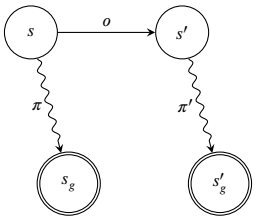


Fig. 3. Illustration for Lemma 12.

given (backward reachable) state $s \in S_{\text{bw}}$ the sum $\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) - \sum_{f \in \pi[\![s]\!]} \mathtt{P}(f)$ evaluates to exactly the same value for all s-plans $\pi = \langle o_1, \ldots, o_n \rangle$.

Lemma 10. *Let* $\mathtt{Q}$ *denote an operator-potential function for* $\mathtt{P}$*, let* $s \in S_{\text{bw}}$ *denote a backward reachable state, let* $s_g \supseteq G$ *and* $s'_g \supseteq G$ *denote two goal states, and let* $\pi = \langle o_1, \ldots, o_n \rangle \in \mathcal{E}_{\text{bw}}$ *and* $\pi' = \langle o'_1, \ldots, o'_m \rangle \in \mathcal{E}_{\text{bw}}$ *denote two* s*-plans such that* $\pi[\![s]\!] = s_g$ *and* $\pi'[\![s]\!] = s'_g$*. Then*

$$\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) - \sum_{f \in s_g} \mathtt{P}(f) = \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [m]} \mathtt{Q}(o'_i) - \sum_{f \in s'_g} \mathtt{P}(f).$$

Proof. Since $\sum_{f \in I} \mathtt{P}(f)$ appears on both sides of the equation, we need to prove that

$$\sum_{i \in [n]} \mathtt{Q}(o_i) - \sum_{f \in s_g} \mathtt{P}(f) = \sum_{i \in [m]} \mathtt{Q}(o'_i) - \sum_{f \in s'_g} \mathtt{P}(f).$$

Since $\pi[\![s]\!] = s_g$, it follows from Proposition 5 that $\sum_{f \in s} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) = \sum_{f \in s_g} \mathtt{P}(f)$, and similarly for π' and s'_g we have that $\sum_{f \in s} \mathtt{P}(f) + \sum_{i \in [m]} \mathtt{Q}(o'_i) = \sum_{f \in s'_g} \mathtt{P}(f)$. Therefore, it follows that

$$\sum_{i \in [n]} \mathtt{Q}(o_i) - \sum_{f \in s_g} \mathtt{P}(f) = -\sum_{f \in s} \mathtt{P}(f) = \sum_{i \in [m]} \mathtt{Q}(o'_i) - \sum_{f \in s'_g} \mathtt{P}(f),$$

which concludes the proof. □

Next, we show that $h^{\mathtt{Q}}_{\text{bw}}$ is init-aware.

Lemma 11. *Let* $\mathtt{Q}$ *denote an operator-potential function for* $\mathtt{P}$*, let* $\pi = \langle o_1, \ldots, o_n \rangle$ *denote a plan, and let* $s_g = \pi[\![I]\!]$*. Then* $\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) - \sum_{f \in s_g} \mathtt{P}(f) = 0$.

Proof. It follows directly from Proposition 5, because $s_g = \pi[\![I]\!]$ and therefore $\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i) = \sum_{f \in s_g} \mathtt{P}(f)$. □

In the following Lemma 12 we show that $h^{\mathtt{Q}}_{\text{bw}}$ is backward consistent under the assumption that Equation (3) holds for $\mathtt{P}$. It follows from the state-dependency of $h^{\mathtt{Q}}_{\text{bw}}$ and the fact that if Equation (3) holds, then $-\mathtt{Q}(o) \leq \text{cost}(o)$ holds, i.e., transitioning over an operator o cannot decrease the heuristic estimate by more than $\text{cost}(o)$.

Lemma 12. *Let* $\mathtt{Q}$ *denote an operator-potential function for* $\mathtt{P}$*, let* $s, s' \in S_{\text{bw}}$ *and* $o \in \mathcal{O}$ *denote two backward reachable states and an operator such that* $o[\![s]\!] = s'$*, let* $\pi = \langle q_1, \ldots, q_n \rangle \in \mathcal{E}_{\text{bw}}$ *denote an* s*-plan, let* $\pi' = \langle q'_1, \ldots, q'_m \rangle \in \mathcal{E}_{\text{bw}}$ *denote an* s'*-plan, and let* $s_g = \pi[\![s]\!]$ *and* $s'_g = \pi'[\![s']\!]$*. If Equation (3) holds for* $\mathtt{P}$*, then*

$$\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [m]} \mathtt{Q}(q'_i) - \sum_{f \in s'_g} \mathtt{P}(f) \leq \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(q_i) - \sum_{f \in s_g} \mathtt{P}(f) + \text{cost}(o).$$

Proof. Let $\rho = \langle o, q'_1, \ldots, q'_n \rangle$ (see illustration in Fig. 3). From Lemma 10 and the fact that $\rho \in \mathcal{E}_{\text{bw}}$ and it is applicable in s and $\rho[\![s]\!] = s'_g$, it follows that

$$\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(q_i) - \sum_{f \in s_g} \mathtt{P}(f) = \sum_{f \in I} \mathtt{P}(f) + \mathtt{Q}(o) + \sum_{i \in [m]} \mathtt{Q}(q'_i) - \sum_{f \in s'_g} \mathtt{P}(f),$$

therefore

$$\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [m]} \mathtt{Q}(q'_i) - \sum_{f \in s'_g} \mathtt{P}(f) = \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(q_i) - \sum_{f \in s_g} \mathtt{P}(f) - \mathtt{Q}(o).$$

Therefore it is enough to prove that $-\mathtt{Q}(o) \leq \text{cost}(o)$ which follows directly from Definition 3 and Equation (3). □

Algorithm 2: Partitioning of the goal states such that all states within each partition have the same $h^{\mathtt{P}}$-value.

Input: A set of variables $\mathcal{V} = \{V_1, \dots, V_n\}$, a disambiguation map D_G for goal G, a potential function $\mathtt{P}$.
Output: Partitioning P^G of the goal states by their $h^{\mathtt{P}}$-values.

```
1  P^G ← {⟨0, S⟩} where S is a set of all states;
2  for i = 1, ..., n do
3      M ← ∅;
4      for each f ∈ D_G(V_i) do
5          B ← {s ∈ S | f ∈ s};
6          InsertOrUpdate(M, P(f), B) ;
7      P^G ← Merge(P^G, M) ;
8  return P^G;

9  function InsertOrUpdate(M, h, B)
10     if there exists ⟨h, B'⟩ ∈ M then
11         M ← (M \ {⟨h, B'⟩}) ∪ {⟨h, B' ∪ B⟩};
12     else
13         M ← M ∪ {⟨h, B⟩};

14 function Merge(M, M')
15     X ← ∅;
16     for each ⟨h, B⟩ ∈ M do
17         for each ⟨h', B'⟩ ∈ M' do
18             InsertOrUpdate(X, h + h', B ∩ B');
19     return X;
```

Now we are ready to prove that goal-corrected operator-potential backward heuristics are well-defined (i.e., state-dependent), init-aware, backward consistent, and therefore also backward admissible.

Theorem 13. *$h^{\mathtt{Q}}_{\mathrm{bw}}$ is well-defined, init-aware, backward consistent, and backward admissible.*

Proof. It follows from Lemma 10 that $h^{\mathtt{Q}}_{\mathrm{bw}}$ is well-defined because given any backward reachable state s, the value of $h^{\mathtt{Q}}_{\mathrm{bw}}(s)$ is the same for all s-plans. Init-awareness follows directly from Lemma 11, backward consistency follows directly from Lemma 12, and backward admissibility follows from init-awareness and backward consistency. □

3.3. Partitioning of goal states into BDDs for backward search

Backward search starts in goal states and proceeds towards the initial state. So, given a state s reached during the backward search, backward heuristics estimate the cost of the optimal I-s-path. Incorporating heuristic values of goal states into Equation (7) allows us to define a backward heuristic that is state-dependent, backward consistent, backward admissible, and at the same time we can associate each operator with the change of the heuristic value it induces. However, it also comes with a price. Goal conditions of planning tasks are partial states, so they can define an exponential number of goal states. It may seem we need to enumerate all (forward reachable) goal states in order to compute heuristic values for them, which would be in general infeasible. Fortunately, in symbolic search, sets of states are represented as BDDs whose size can be exponentially smaller than the number of states they represent. So, in order to use $h^{\mathtt{Q}}_{\mathrm{bw}}$ in the context of symbolic search, we do not need to enumerate all forward reachable goal states, but rather partition those goal states into multiple BDDs so that each BDD represents all forward reachable goal states with the same $h^{\mathtt{P}}$-value (or an overapproximation of them). Algorithm 2 encapsulates the algorithm that does exactly that.

The main idea behind Algorithm 2 is as follows. Given a fact f, let $\mathcal{S}_f = \{s \in \mathcal{S} \mid f \in s\}$ denote a set of all states containing f. First, it is easy to see that, given a variable $V \in \mathcal{V}$ and its value $v \in \mathrm{dom}(V)$, for every state $s \in \mathcal{S}_{\langle V,v\rangle}$ it holds that $s[V] = v$ and therefore $\mathtt{P}(\langle V, s[V]\rangle) = \mathtt{P}(\langle V, v\rangle)$. So, given a set of distinct variables $V_1, \dots, V_n \in \mathcal{V}$ and their respective values $v_1 \in \mathrm{dom}(V_1), \dots, v_n \in \mathrm{dom}(V_n)$, for every state $s \in \bigcap_{i\in[n]} \mathcal{S}_{\langle V_i,v_i\rangle}$ it holds that $s[V_i] = v_i$ for every $i \in [n]$, and therefore $\sum_{i\in[n]} \mathtt{P}(\langle V_i, s[V_i]\rangle) = \sum_{i\in[n]} \mathtt{P}(\langle V_i, v_i\rangle)$. In other words, starting from sets of states $\mathcal{S}_{f_1}, \dots, \mathcal{S}_{f_n}$ where each f_i is from a different variable V_i, we can construct a set of more specific states by taking the intersection between sets $\mathcal{S}_{f_i}$ while keeping track of the sum of potentials over variables V_i.

Second, given a variable V and two values $v, v' \in \mathrm{dom}(V)$, for every state $s \in \mathcal{S}_{\langle V,v\rangle} \cup \mathcal{S}_{\langle V,v'\rangle}$ it holds that $s[V] = v$ or $s[V] = v'$. So, if $\mathtt{P}(\langle V, v\rangle) = \mathtt{P}(\langle V, v'\rangle)$, then also $\mathtt{P}(\langle V, s[V]\rangle) = \mathtt{P}(\langle V, v\rangle)$ for every state $s \in \mathcal{S}_{\langle V,v\rangle} \cup \mathcal{S}_{\langle V,v'\rangle}$. And we can generalize this idea to a set of distinct variables $V_1, \dots, V_n \in \mathcal{V}$ and their values $v_1, v'_1 \in \mathrm{dom}(V_1), \dots, v_n, v'_n \in \mathrm{dom}(V_n)$: If $\sum_{i\in[n]} \mathtt{P}(\langle V_i, v_i\rangle) = \sum_{i\in[n]} \mathtt{P}(\langle V_i, v'_i\rangle)$, then for every state $s \in \bigcap_{i\in[n]} \mathcal{S}_{\langle V_i,v_i\rangle} \cup \bigcap_{i\in[n]} \mathcal{S}_{\langle V_i,v'_i\rangle}$ it holds that $s[V_i] = v_i$ or $s[V_i] = v'_i$ for every $i \in [n]$, and therefore also $\sum_{i\in[n]} \mathtt{P}(\langle V_i, s[V_i]\rangle) = \sum_{i\in[n]} \mathtt{P}(\langle V_i, v_i\rangle)$.

Algorithm 2 puts these two ideas together. It iterates over all variables one by one (outer cycle on lines 2 to 7). For each variable V_i, it considers only the values of V_i that can be part of some (forward reachable) goal state (line 4), and partitions all states having those values by their potential values (lines 3 to 6). Finally, it merges the partitioning over the variable V_i into the partitioning over the variables $V_1, \dots, V_{i-1}$ achieved in the previous step while keeping track of the sum of potentials over the variables $V_1, \dots, V_i$.

Since only the facts that can be part of a goal state are considered, the resulting partitioning P^G is a partitioning of goal states only (Theorem 14 (A1)). Since the disambiguation map D_G overapproximates forward reachable states by definition, P^G contains all forward reachable goal states (Theorem 14 (A2)). Since the function `Merge` is always called only for a partitioning M over variables $V_1, \dots, V_{i-1}$ and a partitioning M' over the variable V_i, taking the intersections on line 18 must result in a partitioning over variables $V_1, \dots, V_i$ eventually terminating with the partitioning over all variables (Theorem 14 (A3)). And finally, since, on line 18, h is the sum of potentials over variables $V_1, \dots, V_{i-1}$, and h' is the potential over the variable V_i, then $h + h'$ is the sum of potentials over variables $V_1, \dots, V_i$ which eventually results in the sum over all variables, i.e., the value of the potential heuristic (Theorem 14 (A4)).

Theorem 14. *Let $\mathcal{V} = \{V_1, \dots, V_n\}$, D_G, and $\mathtt{P}$ denote inputs of Algorithm 2, and let $P^G = \{\langle h_1, P^G_1\rangle, \dots, \langle h_m, P^G_m\rangle\}$ denote the output of Algorithm 2. Then*

A1. *for every $s \in \bigcup_{j\in[m]} P^G_j$ it holds that $G \subseteq s$, i.e., P^G contains only goal states and nothing else; and*
A2. *for every forward-reachable goal state s_g it holds that $s_g \in \bigcup_{j\in[m]} P^G_j$, i.e., P^G contains all forward-reachable goal states; and*
A3. *for every $j, k \in [m]$ such that $j \neq k$ it holds that $P^G_j \cap P^G_k = \emptyset$ and $h_j \neq h_k$, i.e., P^G is, indeed, a partitioning; and*
A4. *for every $j \in [m]$ and every $s \in P^G_j$ it holds that $h_j = h^{\mathtt{P}}(s)$, i.e., Algorithm 2 partitions goal states based on their $h^{\mathtt{P}}$-values.*

Proof. Given a set of variables $X \subseteq \mathcal{V}$ and a partial state s, let $s|_X$ denote a restriction of s to X, i.e., $s|_X = \{\langle V, v\rangle \mid \langle V, v\rangle \in s, V \in X\}$; and given a set of partial states S, let $S|_X = \{s|_X \mid s \in S\}$.

We start with four invariants that hold in every cycle i of the outer loop (lines 2-7) with respect to the construction of the set M (constructed on lines 3-6):

I1. For every $\langle h, B\rangle \in M$ and every $s \in B$ it holds that $\langle V_i, s[V_i]\rangle \in D_G(V_i)$. This follows from the fact that B is built from the union of sets of states $\{s \in \mathcal{S} \mid f \in s\}$ where $f \in D_G(V_i)$.
I2. For every $\langle h, B\rangle \in M$ and every $s \in B$ it holds that $h = \mathtt{P}(\langle V_i, s[V_i]\rangle)$. This holds because `InsertOrUpdate()` inserts a set of states $B = \{s \in \mathcal{S} \mid f \in s\}$ with $h = \mathtt{P}(f)$ only if the value h is not yet in M, and it replaces $\langle h, B\rangle$ with $\langle h, B \cup B'\rangle$ where $B' = \{s \in \mathcal{S} \mid f' \in s\}$ only if $\mathtt{P}(f') = h$.
I3. For every $\langle h, B\rangle \in M$ it holds that $B|_{\mathcal{V}\setminus\{V_i\}} = \mathcal{S}|_{\mathcal{V}\setminus\{V_i\}}$, i.e., B restricted to all variables excluding V_i is a set of all syntactic partial states over $\mathcal{V} \setminus \{V_i\}$. This follows from the fact that every B on line 5 is constructed so that $B|_{\mathcal{V}\setminus\{V_i\}} = \mathcal{S}|_{\mathcal{V}\setminus\{V_i\}}$ and therefore for every union X of such sets it also holds that $X|_{\mathcal{V}\setminus\{V_i\}} = \mathcal{S}|_{\mathcal{V}\setminus\{V_i\}}$.
I4. For every $\langle h, B\rangle, \langle h', B'\rangle \in M$ such that $\langle h, B\rangle \neq \langle h', B'\rangle$ it holds that $B \cap B' = \emptyset$ and $h \neq h'$. Since `InsertOrUpdate()` maintains the set M so that there no two elements with the same h-value, we have that $h \neq h'$. Since $\{s \in \mathcal{S} \mid f \in s\} \cap \{s \in \mathcal{S} \mid f' \in s\} = \emptyset$ whenever $f \neq f'$, we have that $B \cap B' = \emptyset$.

Since P^G is initialized with the set of all (syntactic) states $\mathcal{S}$ and, in the function `Merge`, the sets of states are constructed only using intersections, it follows from I3 that in every cycle i of the outer loop, the function `Merge` is called on line 7 with the argument P^G such that for every $\langle h, B\rangle \in P^G$ it holds that $B|_{\{V_i,\dots,V_m\}} = \mathcal{S}|_{\{V_i,\dots,V_m\}}$. Therefore, the function `Merge` returns P^G such that for every $\langle h, B\rangle \in P^G$ it holds that $B|_{\{V_{i+1},\dots,V_m\}} = \mathcal{S}|_{\{V_{i+1},\dots,V_m\}}$, and furthermore from I1 it follows that for every $s \in B$ and every $V \in \{V_1, \dots, V_i\}$, it holds that $\langle V, s[V]\rangle \in D_G(V)$. Therefore, at the end of the algorithm, for every $\langle h, B\rangle \in P^G$ and every $s \in B$ and every $V \in \mathcal{V}$, it holds that $\langle V, s[V]\rangle \in D_G(V)$. Therefore, it follows from the definition of D_G that A1 holds because $D_G(V) = \{\langle V, v\rangle\}$ for every $\langle V, v\rangle \in G$, and also A2 holds because we considered all possible facts that can appear in any forward reachable goal state.

From I4 and the fact that `Merge()` uses only intersections to construct sets of states, it follows that $P^G_j \cap P^G_k = \emptyset$ for every $j, k \in [m]$ s.t. $j \neq k$. And since `InsertOrUpdate()` makes sure that the output set does not contain two elements with the same h-value, it follows that A3 holds.

Finally, from I2 and the fact that `Merge` in cycle i sums h and h' such that h is a sum of potentials over variables $V_1, \dots, V_{i-1}$ and h' is the potential over variable V_i, it follows that A4 holds. □

Note that all sets of states can be represented as BDDs, and union ($\cup$) and intersection ($\cap$) between sets of states can be computed as a disjunction ($\vee$) and conjunction ($\wedge$) between BDDs.

Also note that Algorithm 2 can be easily used for generating a symbolic pattern database equivalent to the potential heuristic. Running Algorithm 2 with empty G as the input will produce a partitioning of all states (i.e., all states extending $\emptyset$) by their $h^{\mathtt{P}}$-values. Such symbolic pattern database can be used directly in the variant of BDDA* introduced by Kissmann & Edelkamp [34] (discussed in Section 2.2). However, there are two main reasons why not to use such symbolic pattern databases. First, the computation of the partitioning can be easily infeasible in practice because there is no guarantee that the resulting BDDs will concisely represent the underlying set of states (i.e., in the worst case the size of the BDD can be linear in the number of states it represents, therefore it can grow exponentially). This guarantee does not exist even for the partitioning of goal states and we will focus on this aspect in our experimental evaluation in Section 5.4. Second, and more importantly, symbolic pattern databases generated with Algorithm 2 cannot be more informative than the corresponding potential heuristic which, in turn, is equivalent to the operator-potential heuristics $h^{\mathtt{Q}}_{\text{fw}}$ (and to $h^{\mathtt{Q}}_{\text{bw}}$ on forward reachable states). As we discuss in the next section, $h^{\mathtt{Q}}_{\text{fw}}$ and $h^{\mathtt{Q}}_{\text{bw}}$ can be applied in the symbolic search by a straightforward integration of the underlying operator-potential function $\mathtt{Q}$ into the GHSETA* search, which makes the computation of heuristic values using symbolic pattern databases much more expensive than using GHSETA* with $h^{\mathtt{Q}}_{\text{fw}}$ or $h^{\mathtt{Q}}_{\text{bw}}$ instead.

4. Symbolic search with operator-potential heuristics

The integration of the operator-potential *forward* heuristic in the forward GHSETA* is straightforward. The operator-potential forward heuristic $h^{\mathtt{Q}}_{\text{fw}}$ is defined as $h^{\mathtt{Q}}_{\text{fw}}(s) = \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \mathtt{Q}(o_i)$ (see Equation (5)) and therefore we have that $h^{\mathtt{Q}}_{\text{fw}}(s) = h^{\mathtt{P}}(I) + \sum_{i \in [n]} \mathtt{Q}(o_i)$ (because $h^{\mathtt{P}}(I) = h^{\mathtt{P}}_{\text{fw}}(I) = \sum_{f \in I} \mathtt{P}(f)$). Moreover, we have shown in Theorem 7 that $h^{\mathtt{Q}}_{\text{fw}}$ is forward consistent and forward admissible (assuming the underlying $h^{\mathtt{P}}_{\text{fw}}$ is forward consistent and forward admissible). Therefore, we can integrate $h^{\mathtt{Q}}_{\text{fw}}$ into GHSETA* described in Algorithm 1 by simply setting h_I to $h^{\mathtt{P}}(I)$ and using $\mathtt{Q}$ as the δ_h function. That is, we set the heuristic value of the initial state to $h^{\mathtt{P}}(I)$, and partition operators by their costs and $\mathtt{Q}(o)$ values.

For the backward direction, we also use $\mathtt{Q}$ as the δ_h function, but on top of that we need to partition the set of goal states using Algorithm 2. That is, we cannot start with the BDD representing the set of all goal states (and initialize h_I to $h^{\mathtt{P}}(I)$) because this could result in a path-dependent inconsistent heuristic. What we need to do instead is to generate the partitioning of the (forward reachable) goal states by their $h^{\mathtt{P}}$-values and initialize the open list accordingly. Let $P^G = \{P^G_{h_1}, \dots, P^G_{h_n}\}$ denote all partitions returned by Algorithm 2, where $h_1, \dots, h_n$ are all distinct $h^{\mathtt{P}}$-values goal states can have, i.e., for every $P^G_{h_i} \in P^G$ and every state $s_g \in P^G_{h_i}$ it holds that $h^{\mathtt{P}}(s_g) = h_i$. Moreover, let $h^I_i = h^{\mathtt{P}}(I) - h_i$ for every $i \in [n]$. Then we need to initialize the open list (lines 3 and 4 in Algorithm 1) as $\{\langle \max(0, h^I_i), S_{0,h^I_i} \rangle \mid P^G_{h_i} \in P^G, S_{0,h^I_i} = P^G_{h_i}\}$, i.e., we insert every partition $P^G_{h_i}$ into the open list with the g-value set to zero, h-value set to $h^I_i = h^{\mathtt{P}}(I) - h_i$, and f-value set to $\max(0, h^I_i)$. This is all that is needed, because it ensures that any sequence of operators $\langle o_1, \dots, o_m \rangle$ applied on any state from any S_{0,h^I_i} results in the h-value $h^I_i + \sum_{j \in [m]} \mathtt{Q}(o_j) = h^{\mathtt{P}}(I) - h_i + \sum_{j \in [m]} \mathtt{Q}(o_j)$ which is exactly what we need in order to obtain a goal-corrected operator-potential backward heuristic according to Definition 9 that is backward consistent and backward admissible (Theorem 13).

The bi-directional GHSETA* combines the aforementioned approaches and therefore we can use different operator-potential heuristics in each direction or choose to use blind symbolic search in one direction and an operator-potential heuristic in the other.

5. Experimental evaluation

The proposed heuristics and the GHSETA* algorithm was implemented in C as a part of the cpddl planning library.[3] The inference of potential and operator-potential functions was implemented using CPLEX LP/MIP solver v22.1.0. For the manipulation of BDDs we used the CUDD library v3.0.0.

The translation from PDDL to FDR uses the inference of lifted mutex groups proposed by Fišer [19] that are subsequently used for the creation of FDR variables. Operators and facts are pruned with the h^2 heuristic in forward and backward direction [4], and we used mutex pairs from the forward h^2 heuristic for disambiguation.

Performing operations on BDDs can sometimes be very time-consuming, which significantly reduces performance of the symbolic search. Therefore, we follow the approach used in previous implementations of symbolic planners by applying various time limits on BDD operations to mitigate their negative effect whenever we can. We use a time limit of 30 seconds for applying mutexes on the BDDs representing goal states. When the time limit is reached in the forward or backward GHSETA*, the search is simply performed without mutexes applied on the goal BDDs. In case of bi-directional GHSETA*, when the time limit is reached, the search in the backward direction is disabled, because it is a strong indication that computing successor states in the backward direction will be very slow or it will require a large amount of memory. We also applied 10 seconds time limit on merging transition relation BDDs, i.e., for each cost and operator-potential value we try to build a single BDD representing all operators in that partition, but if we fail to do that within the time limit, we use the disjunctive partitioning [33,57]. In case of bi-directional GHSETA*, we also turn off backward search once the step in the backward direction takes longer than three minutes (i.e., one tenth of the overall time limit as we describe below). This helps symbolic search to proceed without getting stuck in a fruitless attempt to compute a set of states that cannot be efficiently represented as a BDD. We do not apply the same time limit in the forward direction, because computing successors in the backward direction is usually more time-consuming than in the forward direction (i.e., if it takes long in the forward direction, it will probably take even longer in the backward direction).

The experiments were conducted on a cluster of computing nodes with Intel Xeon Scalable Gold 6146 processors. The time and memory limits were set to 30 minutes and 8 GB, respectively. We used all planning domains from the optimal track of International Planning Competitions (IPCs) from 1998 to 2018 excluding the ones containing conditional effects after translation and those that could not be grounded and pruned with h^2 within the time and memory limits. We merged, for each domain, all benchmark suites across different IPCs eliminating duplicate instances, resulting in a total of 1648 planning tasks across 48 domains.[4]

Potential and operator-potential functions were inferred with the following optimization criteria:

- `I`: maximize the heuristic value of the initial state [40], i.e., we set the optimization criteria of the LP/MIP to maximize $\sum_{f \in I} \mathtt{P}(f)$.
- `A+I`: maximize the heuristic value for the average (syntactic) state while enforcing the maximum heuristic value for the initial state [20,44]. We first compute `I` to obtain the maximal heuristic value for the initial state h_I. Then we extend the LP/MIP with the additional constraint $\sum_{f \in I} \mathtt{P}(f) \geq h_I$, and we maximize the sum

[3] Source code is publicly available at https://gitlab.com/danfis/cpddl.

[4] Dataset is available at https://gitlab.com/danfis/pddl-data.

$$\sum_{\langle V,v\rangle\in\mathcal{F}} \frac{\mathrm{P}(\langle V,v\rangle)}{|\mathrm{dom}(V)|}.$$

- $\mathtt{S}_{1\mathtt{k}}{}^{+}\mathtt{I}$: maximize the average heuristic value for 1 000 states sampled using random walks, while enforcing the maximum heuristic value for the initial state [20,44]. We enforce the maximum heuristic value for the initial state as in $\mathtt{A}^{+}\mathtt{I}$. Then we sample 1 000 states S by random walks starting from the initial state with binomially distributed length of walks centered around the double of the maximum h-value for the initial state. Finally, we set the optimization criteria to the maximization of the heuristic value over states S, i.e., we maximize

$$\frac{1}{|S|}\sum_{s\in S}\sum_{f\in s}\mathrm{P}(f).$$

- $\mathtt{M}_2{}^{+}\mathtt{I}$: maximize the average heuristic value for all reachable states approximated with mutexes while enforcing the maximum heuristic value for the initial state [20]. The maximum h-value for the initial state is enforced as in $\mathtt{A}^{+}\mathtt{I}$ and $\mathtt{S}_{1\mathtt{k}}{}^{+}\mathtt{I}$. The optimization criteria is based on estimating, for each fact $f\in\mathcal{F}$, the number of forward reachable states containing f. The details are described by Fišer et al. [20, Section 5.1] as the optimization criteria $\mathrm{opt}^{k}_{\mathcal{M}}$ which we use for $k=2$.

The blind symbolic search is denoted by $\mathtt{b}$. The forward symbolic search is denoted by $\rightharpoonup$, and the backward symbolic search by $\leftharpoonup$: For example, the blind forward search is denoted by $\overrightarrow{\mathtt{b}}$, the backward search with $h^{\mathtt{Q}}_{\mathrm{bw}}$ optimized for $\mathtt{A}^{+}\mathtt{I}$ is denoted by $\overleftarrow{\mathtt{A}^{+}\mathtt{I}}$, and the bi-directional search with $h^{\mathtt{Q}}_{\mathrm{fw}}$ optimized for $\mathtt{A}^{+}\mathtt{I}$ used in the forward direction, and $h^{\mathtt{Q}}_{\mathrm{bw}}$ optimized for $\mathtt{I}$ in the backward direction is denoted by $\overrightarrow{\mathtt{A}^{+}\mathtt{I}}\text{-}\overleftarrow{\mathtt{I}}$. For the blind bi-directional symbolic search, we use the shorthand $\overleftrightarrow{\mathtt{b}}$.

For symbolic search with operator-potential heuristics ($h^{\mathtt{Q}}_{\mathrm{fw}}$ and $h^{\mathtt{Q}}_{\mathrm{bw}}$), we transformed planning tasks so that $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every operator o by the "multiplication" method described in Section 2 using the h^2 mutexes for disambiguation. We also compare to the variant where tasks are transformed to TNF using the (polynomial) method proposed by Pommerening & Helmert [38] improved with disambiguations [20]. We show, however, that this method is almost always detrimental to the performance. Note that the transformed planning tasks are not only used for the computation of potential functions, but must also be used for the symbolic search, because the inferred operator-potentials correspond to the operators of the transformed planning task, not the original task. The time spent in the transformation of planning tasks is always counted as a part of the running time.

Besides our implementation of blind symbolic search, we also compare to the following planners:

- A* with potential heuristics using the same optimization criteria used for operator-potential heuristics ($\mathrm{P}_{\mathtt{I}}$, $\mathrm{P}_{\mathtt{A}^{+}\mathtt{I}}$, $\mathrm{P}_{\mathtt{S}_{1\mathtt{k}}{}^{+}\mathtt{I}}$, and $\mathrm{P}_{\mathtt{M}_2{}^{+}\mathtt{I}}$);
- A* with the LM-Cut (`lmc`) heuristic [29];
- A* with the merge-and-shrink (`ms`) heuristic with SCC-DFP merge strategy, non-greedy bisimulation shrink strategy, and the limit of 50 000 states for the resulting abstract transition system [30,48,49];
- the Complementary2 planner (`comp2`) from IPC 2018 [24,25];
- the Scorpion planner (`scrp`) from IPC 2018 [42,43];
- the cGamer planner from IPC 2014 [34,56] with the PDB heuristic (`cgm`), i.e., an implementation of the symbolic search with pattern databases.

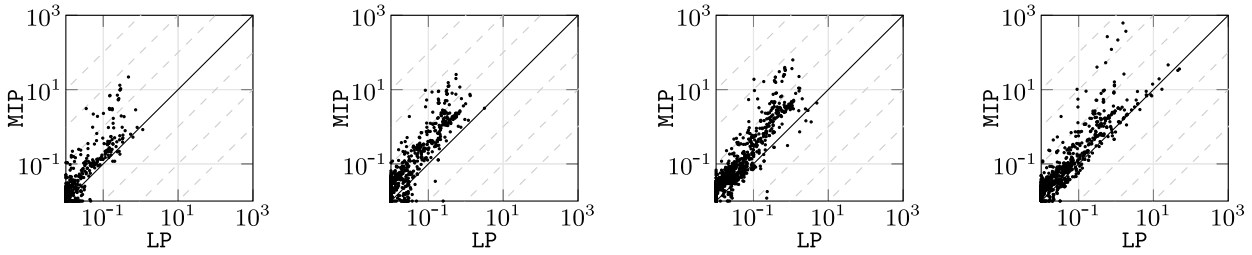
We do not show a detailed comparison to the implementation of blind symbolic search competing in IPC 2011 and 2014 (`smb`), because our implementation has overall better performance. The overall number of solved tasks is 943 by $\overrightarrow{\mathtt{b}}$ in contrast to 852 by $\overrightarrow{\mathtt{smb}}$, 795 by $\overleftarrow{\mathtt{b}}$ vs. 702 by $\overleftarrow{\mathtt{smb}}$, and 1 055 by $\overleftrightarrow{\mathtt{b}}$ vs. 942 by $\overleftrightarrow{\mathtt{smb}}$. Moreover, there are only 27, 12, and 13 individual tasks solved by $\overrightarrow{\mathtt{smb}}$, $\overleftarrow{\mathtt{smb}}$, and $\overleftrightarrow{\mathtt{smb}}$ that are not solved by $\overrightarrow{\mathtt{b}}$, $\overleftarrow{\mathtt{b}}$, and $\overleftrightarrow{\mathtt{b}}$, respectively.

The `cgm` planner is compared only on subsets of domains because of its limited support of PDDL features like conditional effects, inequality preconditions, or quantifiers (we had to exclude domains caldera, cavediving, GED, maintenance, movie, mprime, snake, spider, termes, and trucks).

5.1. Operator-potential functions via mixed-integer linear programs

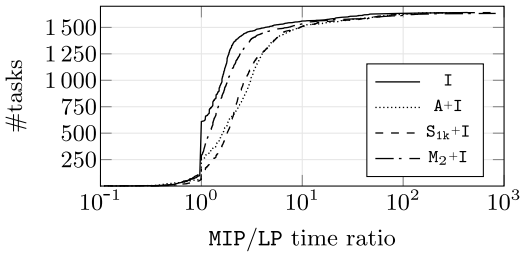
Potential functions (for state-space search) are typically inferred using linear programs (LPs). Nevertheless, a smooth integration of operator-potential heuristics in the symbolic search requires integer-valued operator-potentials which in turn requires solving mixed-integer linear programs (MIPs).

Although there always exists an operator-potential heuristic (e.g., assigning zero to all operators), solving MIP instead of LP may result in a less informative heuristic because the MIP used for operator-potentials is more restricted than the corresponding LP for (fact) potentials. To get a sense of how much does using integer-valued operator-potentials cost in terms of a loss of informativeness, we focus on the potential heuristic optimized for the initial state ($\mathtt{I}$) which gives us the highest possible estimate for the initial state. Comparing how its values for initial states change if the MIP is used instead of LP shows that it actually almost never changes. We found that we get a smaller heuristic value in only 17 tasks from four domains (two tasks in nomystery, three in pegsol, eight in pipesworld-notankage, and four in pipesworld-tankage), and the heuristic values always differ only by one. So, using MIP instead of LP almost never leads to a loss of informativeness. However, solving a MIP, which is NP-complete, is typically much more time and memory demanding than solving a LP, which can be done in polynomial time.

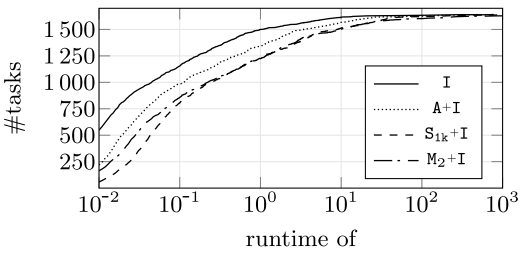


(a) I (b) A+I (c) S_{1k}+I (d) M_2+I

Fig. 4. Per-task comparison of the time in seconds needed for solving LP formulations of potential functions (on horizontal axis), and the corresponding MIP formulations of operator-potential functions (on vertical axis).



(a) Cumulative number of tasks on the vertical axis over the ratio $r_{\texttt{MIP}}/r_{\texttt{LP}}$ on the horizontal axis, where $r_{\texttt{MIP}}$ is the runtime of the MIP variant, and $r_{\texttt{LP}}$ is the runtime of the LP variant.



(b) Cumulative number of tasks on the vertical axis over the absolute runtime in seconds of the MIP variant on the horizontal axis.

Fig. 5. Cumulative graphs comparing solving MIP and LP variants of (operator-) potential heuristics. Only tasks where both MIP and LP was solved within time and memory limits are considered.

Fig. 4 shows per-task comparisons of the runtime of LP and MIP solvers for different variants of (operator-) potential heuristics. Note that I requires solving one LP (or MIP), whereas A+I, M_2+I, and S_{1k}+I require solving two LPs (MIPs)—the first one for getting maximal heuristic value for the initial state which is then used in the second one as an additional constraint. Although solving the MIP is indeed almost always slower (sometimes by more than two orders of magnitude), the runtime is not a significant limiting factor in most tasks. This can be observed in Fig. 5a depicting a cumulative number of tasks with successfully inferred operator-potentials on y axis versus the ratio between the runtime of MIP and LP variants on x axis, i.e., the point (x, y) corresponds to y tasks where the ratio between the runtime of MIP over LP is x or less. For all tested optimization criteria of potential heuristics, the slowdown is well below the factor of ten for most of the tasks, and the median slowdown is 1.4 for I, 2.6 for A+I, 2.3 for S_{1k}+I, and 1.6 for M_2+I.

Fig. 5b depicts the runtime in absolute numbers as a cumulative graph of the number of tasks over the runtime of the MIP variant. It shows that the operator-potential function can be found within one second for most tasks, and under ten seconds for almost all tasks. The runtime higher than ten seconds occurred in only 25, 74, 127, and 130 tasks for I, A+I, S_{1k}+I, and M_2+I, respectively. In contrast to the LP variant, MIP could not be solved within the time or memory limit in only one task from the caldera domain, four from pipesworld-tankage, and two from spider for I and A+I, and additionally three more tasks from airport and four more from pipesworld-notankage for S_{1k}+I and M_2+I. Overall, using MIP instead of LP is rarely a bottleneck, primarily because it is computed only once before the search starts.

5.2. Normalization of planning tasks

State-dependent and (forward and backward) consistent heuristics $h^{\mathtt{Q}}_{\text{fw}}$ and $h^{\mathtt{Q}}_{\text{bw}}$ require that $\text{vars}(\text{pre}(o)) = \text{vars}(\text{eff}(o))$ for every operator $o \in \mathcal{O}$. In the set of benchmarks we use here, this is the case in 485 out of 1 648 tasks. The rest of the tasks has to be transformed to this form. As already described in Section 2, it can be done either by the polynomial method described by Pommerening & Helmert [38] and Fišer et al. [20], denoted by poly, or by the (more brute-force) "multiplication" method (mult). The disadvantage of the poly method is that it can introduce many zero-cost operators, and the disadvantage of the mult method is that it can incur in an exponential blow-up of the number of operators. Nevertheless, Fig. 6 shows that it rarely happens that the number of operators is significantly increased by mult in our dataset. In fact, only one task (from the caldera domain) could not be transformed by mult due to the memory limit, and the number of operators grew more than two-fold in only two domains: In maintenance, the number of operators was between 2.3 and 2.7 times higher with mult. In agricola, the number of operators increased 7- to 16-fold.

The runtime also does not seem to be an issue. The transformation takes more than one second in only 28 tasks for mult in contrast to 34 tasks for poly. The maximum runtime of mult is 6.7 seconds in contrast to 9 seconds of poly, and the median of the ratio between the runtimes of mult and poly is one. Therefore, both methods are about as fast as the other.

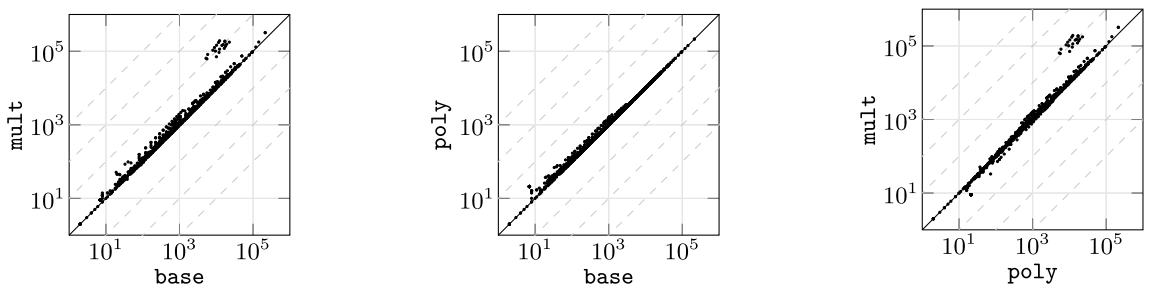


Fig. 6. Per-task comparison of the number of operators before and after the transformation using `poly` or `mult`; `base` denotes the number of operators in the original planning task (before the transformation).

Table 1
Comparison of the `poly` and `mult` methods in terms of the number of domains where one method solved more tasks than the other, the number of tasks solved by one method but not the other, and the overall number of solved tasks.

		$\overrightarrow{\mathtt{I}}$	$\overrightarrow{\mathtt{A^+I}}$	$\overrightarrow{\mathtt{S^+_{1k}I}}$	$\overrightarrow{\mathtt{M^+_2I}}$	$\overleftarrow{\mathtt{I}}$	$\overleftarrow{\mathtt{A^+I}}$	$\overleftarrow{\mathtt{S^+_{1k}I}}$	$\overleftarrow{\mathtt{M^+_2I}}$	$\overrightarrow{\mathtt{A^+I}}$-$\overleftarrow{\mathtt{b}}$	$\overrightarrow{\mathtt{A^+I}}$-$\tilde{\mathtt{I}}$
`poly`	#domains with higher coverage than `mult`	0	0	0	0	1	1	2	1	0	0
	#tasks solved by `poly` but not by `mult`	0	1	1	1	5	1	4	1	1	1
	overall number of solved tasks	884	998	980	990	649	610	616	610	827	818
`mult`	#domains with higher coverage than `poly`	21	18	18	19	32	31	26	31	37	39
	#tasks solved by `mult` but not by `poly`	108	120	121	124	174	185	158	184	337	304
	overall number of solved tasks	992	1 117	1 100	1 113	818	794	770	793	1 163	1 121

Table 2
"Domain Dominance": the row x and column y shows the number of domains where the method x solved more tasks than the method y. "Task Dominance": the row x and column y shows the number of tasks solved by x but not by y. "tot": the overall number of solved tasks (coverage). The number in the cell (x, y) is in bold if it is higher than the number in (y, x). The most interesting numbers are highlighted with grey background: We highlight comparisons between GHSETA* and A* with the same (operator-) potential heuristics, and we highlight comparisons between the forward blind symbolic search ($\overrightarrow{\mathtt{b}}$) and the forward GHSETA* with $h^{\mathtt{q}}_{\mathtt{fw}}$.

	Domain Dominance									Task Dominance									
	$\overrightarrow{\mathtt{A^+I}}$	$\overrightarrow{\mathtt{M^+_2I}}$	$\overrightarrow{\mathtt{S^+_{1k}I}}$	$\mathtt{P_{A^+I}}$	$\mathtt{P_{M^+_2I}}$	$\mathtt{P_{S^+_{1k}I}}$	$\overleftarrow{\mathtt{I}}$	$\overleftarrow{\mathtt{b}}$	$\mathtt{P_I}$	$\overrightarrow{\mathtt{A^+I}}$	$\overrightarrow{\mathtt{M^+_2I}}$	$\overrightarrow{\mathtt{S^+_{1k}I}}$	$\mathtt{P_{A^+I}}$	$\mathtt{P_{M^+_2I}}$	$\mathtt{P_{S^+_{1k}I}}$	$\overleftarrow{\mathtt{I}}$	$\overleftarrow{\mathtt{b}}$	$\mathtt{P_I}$	tot
$\overrightarrow{\mathtt{A^+I}}$	–	**4**	**13**	**25**	**25**	**25**	**30**	**31**	**31**	–	**6**	**21**	**149**	**151**	**153**	**125**	**184**	**200**	1 117
$\overrightarrow{\mathtt{M^+_2I}}$	1	–	**11**	**25**	**25**	**25**	**30**	**31**	**31**	2	–	**18**	**151**	**151**	**155**	**121**	**180**	**201**	1 113
$\overrightarrow{\mathtt{S^+_{1k}I}}$	2	4	–	**22**	**22**	**21**	**26**	**30**	**27**	4	5	–	**140**	**141**	**142**	**110**	**170**	**189**	1 100
$\mathtt{P_{A^+I}}$	9	9	12	–	**3**	**4**	**19**	**24**	**16**	36	42	44	–	**5**	**14**	**113**	**169**	**65**	1 004
$\mathtt{P_{M^+_2I}}$	8	9	12	0	–	3	**19**	**24**	**16**	33	37	40	0	–	**13**	**108**	**164**	**61**	999
$\mathtt{P_{S^+_{1k}I}}$	9	10	12	2	**4**	–	**20**	**24**	**17**	32	38	38	6	10	–	**107**	**162**	**69**	996
$\overleftarrow{\mathtt{I}}$	0	0	0	14	14	14	–	**21**	**18**	0	0	2	101	101	103	–	**76**	**134**	992
$\overleftarrow{\mathtt{b}}$	4	4	5	16	16	17	7	–	**20**	10	10	13	108	108	109	27	–	**132**	943
$\mathtt{P_I}$	5	6	7	0	0	3	16	19	–	22	27	28	0	1	12	81	128	–	939

So, neither of the methods seem to be detrimental in terms of the size of the resulting planning task or the runtime overhead they incur. However, it turns out that the `mult` method has a much better synergy with the symbolic search with operator-potential heuristics than `poly`, as can be observed in Table 1. For example, there is only a single task from the whole dataset where using `poly` is beneficial over using `mult` in the forward search—it is the one task from the caldera domain mentioned above, where `mult` could not successfully transform the planning task within the memory limit. We think the clear superiority of `mult` is caused by the auxiliary zero-cost operators created by `poly`, because a single operator in the task created with `mult` may correspond to a sequence of operators in the task created with `poly`. Thus the change of the heuristic value induced by such an operator is dissolved into multiple operators for `poly`. For these reasons, in all of the following experiments, we consider the transformation method `mult` only.

5.3. Forward search

As we discussed in Section 5.1, operator-potential heuristics tend to retain informativeness of the corresponding potential heuristics. So, the next question is whether the information provided by operator-potential heuristics increases the efficiency of the symbolic search. Table 2 compares all variants of GHSETA*, state-space search A* with the same potential heuristics, and the blind forward and bi-directional symbolic search. GHSETA* variants are clearly superior to their A* counterparts in overall numbers—be it the overall number of solved tasks, number of domains in which GHSETA* dominates A*, or the number of tasks solved by GHSETA* but not A*. However, we can still observe some complementarity between the methods (in particular, for the potentials optimized for the initial state (`I`)).

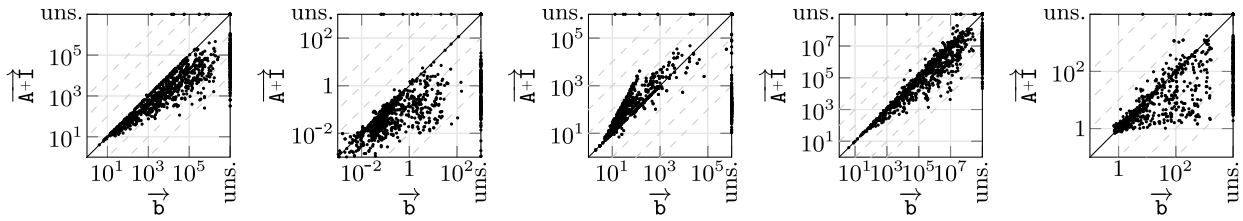
Table 3
Per-domain comparison of the number of solved tasks for the forward GHSETA* with h^{0}_{fw}, A* with potential heuristics, and forward and bi-directional blind symbolic search. The row "others" sums over domains with exactly the same number of solved tasks by all compared methods. "+" indicates that GHSETA* solved every task that was solved by A* with the same potential heuristic; "○" indicates that GHSETA* solved every task solved by $\overrightarrow{\mathrm{b}}$; and "⊕" indicates the combination of both + and ○ occurring at the same time. Finally, we highlight in blue (⊕) the cases where GHSETA* has strictly more coverage than any of the two methods it combines. (For interpretation of the colors in the table, the reader is referred to the web version of this article.)

domain	$\overrightarrow{\mathrm{I}}$	$\overrightarrow{\mathrm{A^+I}}$	$\overrightarrow{\mathrm{S_{1k}{}^+I}}$	$\overrightarrow{\mathrm{M_2{}^+I}}$	$\mathrm{P_I}$	$\mathrm{P_{A^+I}}$	$\mathrm{P_{S_{1k}{}^+I}}$	$\mathrm{P_{M_2{}^+I}}$	$\overrightarrow{\mathrm{b}}$	$\overleftrightarrow{\mathrm{b}}$
agricola (20)	+ 13	⊕ **20**	⊕ **20**	⊕ **20**	3	3	3	3	**20**	**20**
airport (30)	○ 27	○ 27	○ 27	○ 27	**30**	**30**	**30**	**30**	26	27
barman (34)	+ 11	+ 16	+ 16	+ 16	11	11	11	11	**18**	**18**
blocks (35)	⊕ 23	⊕ 31	⊕ 30	⊕ 31	21	28	28	28	21	**33**
caldera (20)	**17**	**17**	**17**	**17**	12	12	12	12	**17**	**17**
childsnack (20)	⊕ 4	⊕ **5**	⊕ **5**	⊕ **5**	0	0	0	0	4	4
data-network (20)	+ 9	⊕ **13**	+ 9	⊕ **13**	9	9	9	9	11	**13**
depot (22)	6	⊕ **11**	○ 10	⊕ **11**	9	**11**	**11**	**11**	7	8
driverlog (20)	⊕ 13	⊕ **14**	⊕ 13	⊕ **14**	13	13	13	13	11	**14**
elevators (50)	⊕ 35	⊕ 35	⊕ 35	⊕ 35	31	31	31	31	35	**43**
floortile (40)	⊕ 17	⊕ 17	⊕ 17	⊕ 17	16	16	16	16	17	**34**
freecell (80)	○ 42	○ 69	○ 68	○ 67	65	**72**	**72**	69	20	25
ged (20)	⊕ 15	⊕ 15	○ 15	⊕ 15	15	15	19	15	15	**20**
gripper (20)	⊕ **20**	⊕ **20**	⊕ **20**	⊕ **20**	8	8	8	8	**20**	**20**
hiking (20)	+ 14	+ 14	+ 15	+ 14	13	14	14	14	16	**18**
logistics (61)	⊕ 27	⊕ **28**	⊕ **28**	⊕ **28**	13	24	24	24	21	25
mprime (35)	⊕ 28	⊕ **30**	⊕ 29	⊕ **30**	24	24	24	24	27	27
mystery (19)	○ 15	⊕ **19**	⊕ **19**	⊕ **19**	16	18	18	18	15	15
nomystery (20)	⊕ 14	⊕ **18**	⊕ 17	⊕ **18**	10	14	14	14	11	**18**
openstacks (100)	⊕ 88	⊕ **91**	⊕ **91**	⊕ **91**	57	57	57	57	87	87
parcprinter (50)	○ 44	○ **48**	⊕ 45	○ **48**	**48**	**48**	45	**48**	41	39
parking (40)	○ 0	○ 13	○ 13	○ 13	11	**16**	**16**	**16**	0	5
pathways (30)	⊕ **5**	⊕ **5**	⊕ **5**	⊕ **5**	4	4	4	4	**5**	**5**
pegsol (50)	⊕ **48**	⊕ **48**	⊕ **48**	⊕ **48**	**48**	**48**	**48**	**48**	46	**48**
petri-net-alignment (20)	9	11	10	10	13	13	11	13	12	**19**
pipesworld-notankage (50)	○ 22	○ 25	○ 25	○ 23	25	**30**	29	29	17	17
pipesworld-tankage (50)	⊕ 18	⊕ **20**	○ **20**	⊕ **20**	16	19	**20**	19	17	17
rovers (40)	⊕ 13	⊕ **14**	⊕ **14**	⊕ **14**	6	8	8	8	13	**14**
satellite (27)	⊕ 7	⊕ **11**	⊕ 10	⊕ **11**	6	6	6	6	7	**11**
scanalyzer (50)	⊕ **23**	⊕ **23**	⊕ **23**	⊕ **23**	**23**	**23**	**23**	**23**	21	21
snake (20)	○ 10	○ 11	○ 11	○ 11	**15**	**15**	**15**	**15**	7	7
sokoban (50)	○ 48	⊕ **50**	⊕ **50**	⊕ **50**	**50**	**50**	**50**	**50**	48	48
spider (20)	○ 11	○ 13	○ 11	○ 12	14	**16**	**16**	15	7	7
storage (30)	⊕ 15	⊕ **16**	⊕ **16**	⊕ **16**	15	**16**	**16**	**16**	15	15
termes (20)	⊕ 12	⊕ 12	⊕ 12	⊕ 12	12	12	12	12	12	**18**
tetris (17)	○ 13	○ 16	○ 16	○ 16	15	**17**	**17**	**17**	9	12
tidybot (40)	○ 30	⊕ **34**	⊕ **34**	⊕ **34**	32	32	32	32	30	29
tpp (30)	⊕ **12**	⊕ **12**	⊕ **12**	⊕ **12**	7	8	8	8	8	8
transport (70)	23	+ 25	23	+ 25	24	24	24	24	27	**34**
trucks (30)	○ 14	⊕ **16**	⊕ 14	⊕ **16**	14	14	14	14	13	13
visitall (40)	○ 22	○ 22	○ 22	○ 22	**30**	**30**	23	**30**	17	18
woodworking (50)	⊕ 40	⊕ 45	⊕ **48**	⊕ 47	19	29	29	29	38	**48**
zenotravel (20)	○ 10	⊕ **12**	⊕ **12**	⊕ **12**	11	11	11	11	9	11
others (118)	**105**	**105**	**105**	**105**	**105**	**105**	**105**	**105**	**105**	**105**
Σ (1648)	992	**1 117**	1 100	1 113	939	1 004	996	999	943	1 055

The more detailed per-domain comparison in Table 3 indicates that A* with a potential heuristic solves more tasks than GHSETA* with the corresponding operator-potential heuristic mostly in domains where $\overrightarrow{\mathrm{b}}$ performs worse than A* with potential heuristics. Nevertheless, GHSETA* performs at least as good as (and usually better than) the corresponding A* with potential heuristics in an overwhelming majority of domains.

The comparison to $\overrightarrow{\mathrm{b}}$ in Table 2 shows that enhancing symbolic search with operator-potential heuristics greatly increases the overall number of solved tasks, and it is rarely detrimental. Table 3 shows a great synergy between the methods across the whole benchmark set. This suggests that the partitioning of operators by their operator-potentials induces a compact representation of sets of states using BDDs which can also be observed in Fig. 7.

Fig. 7a shows that the size of BDDs measured as the number of nodes of the BDDs consistently decreases when the operator-potential heuristic is used, and this, as expected, leads to a speedup per expanded BDD (Fig. 7b) as most operations on BDDs are polynomial in the number of BDD nodes. Fig. 7c shows that the number of expanded BDDs (sets of states) increases, which, again, is expected, because the sets of states during the search are partitioned not only by g-values but also h-values. Nevertheless, the overall search effort is reduced, which can be observed in Fig. 7d comparing the number of BDD nodes from all expanded BDDs, and in Fig. 7e showing the overall runtime in seconds. As the plots show, the number of total BDD nodes across all BDDs involved in the search is rarely significantly increased when using operator-potential heuristics. This shows that operator-potential heuristics can



(a) Average number of BDD nodes per expanded BDD (i.e., average size of a BDD).

(b) Average runtime (in seconds) per expanded BDD.

(c) Number of expanded BDDs (i.e., number of expanded sets of states).

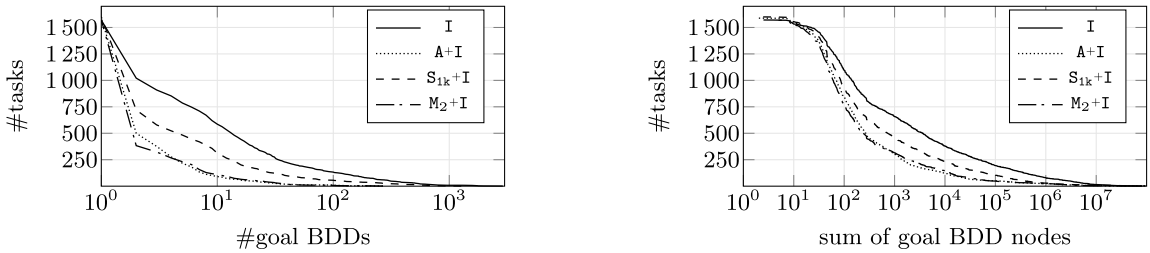
(d) Sum of expanded BDD nodes.

(e) Runtime in seconds.

Fig. 7. Per-task comparison between the best-performing variant of forward GHSETA*.

Table 4
The number of tasks in which the partitioning of goal states using Algorithm 2 failed. The row "others" sums over domains where there is no difference between the compared methods.

domain	$\overleftarrow{I}$	$\overleftarrow{A+I}$	$\overleftarrow{S_{1k}+I}$	$\overleftarrow{M_2+I}$
airport (30)	**1**	0	0	0
childsnack (20)	**4**	0	1	0
data-network (20)	**1**	0	0	0
depot (22)	1	**2**	1	0
mprime (35)	**1**	0	0	0
pathways (30)	**1**	0	0	0
pipesworld-notankage (50)	12	2	2	**18**
pipesworld-tankage (50)	**16**	15	8	12
rovers (40)	**1**	0	0	0
snake (20)	3	6	2	**15**
sokoban (50)	**2**	**2**	1	0
storage (30)	6	**10**	7	0
tetris (17)	**5**	1	2	1
tidybot (40)	**1**	0	0	0
tpp (30)	**17**	15	16	15
others (1164)	**0**	**0**	**0**	**0**
Σ (1648)	**72**	53	40	61



(a) x-axis: The number of sets of goal states (each set represented as a BDD) with different heuristic values. y-axis: The number of tasks having the given number of goal BDDs or more.

(b) x-axis: The sum of BDD nodes over all sets of goal states. y-axis: The number of tasks having the given number of goal BDD nodes or more.

Fig. 8. Cumulative graphs comparing the number of goal BDDs and the number of nodes they consist of.

overcome the limitations of other "more informed" heuristics that do not induce a good BDD representation of sets of states during the search [51]. In terms of runtime, this translates into speed ups of up to several orders of magnitude, while being detrimental in very few cases.

Overall, it seems GHSETA* with operator-potential heuristics tends to get the best from both the symbolic search and the heuristic search with potential heuristics. Furthermore, one can observe that the combination of symbolic search and operator-potential heuristics is often better than the sum of its parts, i.e., in many domains the combination solves every task solved by any of the two techniques it combines, and it achieves a strictly higher coverage than the best of them.

Table 5
"Domain Dominance": the row x and column y shows the number of domains where the method x solved more tasks than the method y. "Task Dominance": the row x and column y shows the number of tasks solved by x but not by y. "tot": the overall number of solved tasks (coverage).

	Domain Dominance					Task Dominance					
	$\overleftarrow{\mathtt{I}}$	$\overleftarrow{\mathtt{b}}$	$\overleftarrow{\mathtt{A}^+\mathtt{I}}$	$\overleftarrow{\mathtt{M}_2{}^+\mathtt{I}}$	$\overleftarrow{\mathtt{S}_{1k}{}^+\mathtt{I}}$	$\overleftarrow{\mathtt{I}}$	$\overleftarrow{\mathtt{b}}$	$\overleftarrow{\mathtt{A}^+\mathtt{I}}$	$\overleftarrow{\mathtt{M}_2{}^+\mathtt{I}}$	$\overleftarrow{\mathtt{S}_{1k}{}^+\mathtt{I}}$	tot
$\overleftarrow{\mathtt{I}}$	–	**22**	**18**	**18**	**23**	–	**86**	**64**	**65**	**78**	818
$\overleftarrow{\mathtt{b}}$	16	–	9	11	**14**	63	–	**42**	**44**	**59**	795
$\overleftarrow{\mathtt{A}^+\mathtt{I}}$	12	**16**	–	**6**	**14**	40	41	–	**15**	**35**	794
$\overleftarrow{\mathtt{M}_2{}^+\mathtt{I}}$	11	**15**	3	–	**13**	40	42	14	–	**38**	793
$\overleftarrow{\mathtt{S}_{1k}{}^+\mathtt{I}}$	7	12	5	5	–	30	34	11	15	–	770

5.4. Backward search

Goal-corrected operator-potential backward heuristics require partitioning of goal states by their heuristic values. Algorithm 2 from Section 3.3 can provide such partitioning as a set of BDDs, but it can have exponential runtime and it can generate an exponential number of partitions in the worst case.

Table 4 shows the number of tasks per domain where Algorithm 2 did not finish partitioning of goal states within the time limit (the memory limit was never an issue). The partitioning is not possible in only a relatively small number of tasks from the IPC domains and it is highly dependent on the domain and operator-potential heuristic. Moreover, only three tasks where the partitioning failed could be solved by some variant of the backward symbolic search. $\overleftarrow{\mathtt{I}}$ failed to compute partitioning of goal states in one task in airport which was solved by $\overleftarrow{\mathtt{A}^+\mathtt{I}}$, $\overleftarrow{\mathtt{S}_{1k}{}^+\mathtt{I}}$, $\overleftarrow{\mathtt{M}_2{}^+\mathtt{I}}$ and $\overleftarrow{\mathtt{b}}$, and one task in tetris which was solved by $\overleftarrow{\mathtt{A}^+\mathtt{I}}$, $\overleftarrow{\mathtt{S}_{1k}{}^+\mathtt{I}}$ and $\overleftarrow{\mathtt{M}_2{}^+\mathtt{I}}$. $\overleftarrow{\mathtt{S}_{1k}{}^+\mathtt{I}}$ failed to determine partitioning in one task from the tpp domain which was solved by $\overleftarrow{\mathtt{I}}$. Overall, partitioning of goal states fails (with very few exceptions) only in tasks that cannot be solved by any variant of backward symbolic search. The median runtime of Algorithm 2 for all variants is about 1 millisecond, and the averages are 8.3, 3.5, 2.6 and 5 seconds for $\overleftarrow{\mathtt{I}}$, $\overleftarrow{\mathtt{A}^+\mathtt{I}}$, $\overleftarrow{\mathtt{S}_{1k}{}^+\mathtt{I}}$ and $\overleftarrow{\mathtt{M}_2{}^+\mathtt{I}}$, respectively. Therefore, running partitioning of goal states does not seem to be a limiting factor in practice.

To see how many partitions we get for a different operator-potential heuristic, i.e., how many different heuristic values goal states have, we plot cumulative graphs in Fig. 8a showing the number of tasks (on y-axis) having at least the number of goal BDD partitions given on x-axis. The median of the number of partitions is 5 for $\mathtt{I}$, and 1 for $\mathtt{A}^+\mathtt{I}$, $\mathtt{S}_{1k}{}^+\mathtt{I}$, $\mathtt{M}_2{}^+\mathtt{I}$. The average is 42.1, 3.8, 17.8 and 3.8 for $\mathtt{I}$, $\mathtt{A}^+\mathtt{I}$, $\mathtt{S}_{1k}{}^+\mathtt{I}$ and $\mathtt{M}_2{}^+\mathtt{I}$, respectively. The number of tasks with 10 (100) or less partitions is 1 007 (1 438) for $\mathtt{I}$, 1 503 (1 580) for $\mathtt{A}^+\mathtt{I}$, 1 311 (1 546) for $\mathtt{S}_{1k}{}^+\mathtt{I}$, and 1 469 (1 564) for $\mathtt{M}_2{}^+\mathtt{I}$. So, in a majority of tasks the partitioning results in a very low number of partitions, it rarely happens that the number of partitions exceeds 100, and the optimization for the initial state ($\mathtt{I}$) tends to generate more partitions than other methods. The number of partitions is also highly domain-dependent, and larger tasks tend to have more goal partitions than smaller tasks from the same domain.

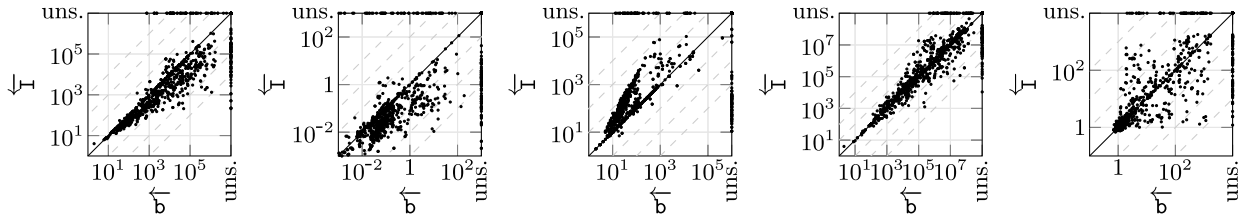
Fig. 8b shows the size of the representation of goal states as a cumulative graph similar to Fig. 8a, i.e., it shows the number of tasks (y-axis) where the sum of the number of BDD nodes over all goal BDDs is at least the number given on the x-axis. Note that the number of goal states in every task is the same for all variants of operator-potential heuristics, but the partitioning of goal states may be different. The graph shows that a higher number of goal BDDs results in a less concise representation of the underlying states which is not surprising. Nevertheless, the difference between the size of representations for different operator-potential heuristics seems to be less profound than the difference between the number of partitions.

Table 5 compares GHSETA* with different variants of $h^{\mathbb{Q}}_{\text{bw}}$ and the blind backward symbolic search in terms of the number of domains where one method solved more tasks than the other ("Domain Dominance"), and the number of tasks solved by one method but not the other ("Task Dominance"). On one hand, we can observe that using operator-potential heuristics $\mathtt{I}$, $\mathtt{A}^+\mathtt{I}$, and $\mathtt{M}_2{}^+\mathtt{I}$ instead of blind search is beneficial in more domains than it is detrimental, $\overleftarrow{\mathtt{I}}$ solves more tasks overall than $\overleftarrow{\mathtt{b}}$, and the overall coverage of $\overleftarrow{\mathtt{b}}$ is almost the same as $\overleftarrow{\mathtt{A}^+\mathtt{I}}$ and $\overleftarrow{\mathtt{M}_2{}^+\mathtt{I}}$. On the other hand, all methods using $h^{\mathbb{Q}}_{\text{bw}}$ seem to be complementary to the blind search and to each other in terms of both domain and task dominance. Nevertheless, $\overleftarrow{\mathtt{I}}$ stands out in this comparison as it not only solves more tasks than any other method, but is also superior to all other methods in the number of dominated domains and tasks. Moreover, we were not able to ascertain any correlation between the number of goal partitions and the number of solved tasks.

Similarly to the forward search, we can also observe in this case that the average size of an expanded BDD is often decreased (Fig. 9a), which leads to a speedup per expanded BDD (Fig. 9b). The number of expanded BDDs is also increased (Fig. 9c) as expected because of the partitioning of states by both g- and h-values. Where the backward search significantly differs from its forward counterpart is the comparison of the size of the BDD representation of sets of states (Fig. 9d) and the overall runtime of backward search (Fig. 9e) which show complementarity of the blind backward search and the backward search with operator-potential heuristic rather than a clear-cut improvement thanks to more informed search as was the case for the forward symbolic search (cf. Fig. 7d and 7e).

5.5. Bi-directional search

Symbolic bi-directional search allows us to combine any variant of the operator-potential heuristics in forward and backward directions. Table 6 shows the comparison of the overall number of solved tasks for all variants of forward and backward GHSETA*



(a) Average number of BDD nodes per expanded BDD (i.e., average size of a BDD).
(b) Average runtime (in seconds) per expanded BDD.
(c) Number of expanded BDDs (i.e., number of expanded sets of states).
(d) Sum of expanded BDD nodes.
(e) Runtime in seconds.

Fig. 9. Per-task comparison between the best-performing variant of backward GHSETA* ($\overleftarrow{\texttt{I}}$) and the backward blind symbolic search ($\overleftarrow{\texttt{b}}$).

Table 6
Overall number of tasks solved by different combinations of forward and backward symbolic search. A value in the row x and the column y is the overall number of solved tasks where x was used for the forward direction and y for the backward direction of the symbolic bi-directional search. ∅ means that no forward or backward search was used. `oracle` refers to selecting the best variant for each task for the respective search direction. The highest coverage of all non-oracle variants is in bold.

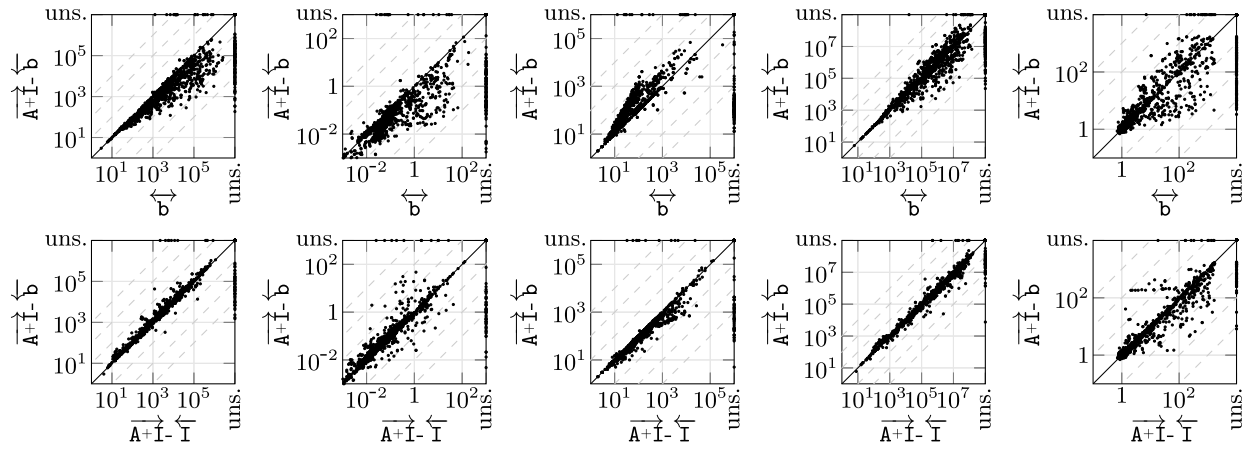
	∅	$\overleftarrow{\texttt{b}}$	$\overleftarrow{\texttt{I}}$	$\overleftarrow{\texttt{A+I}}$	$\overleftarrow{\texttt{M}_2\texttt{+I}}$	$\overleftarrow{\texttt{S}_{1k}\texttt{+I}}$	$\overleftarrow{\texttt{oracle}}$
∅	–	795	818	794	793	770	893
$\overrightarrow{\texttt{b}}$	943	1 055	991	998	998	974	1 084
$\overrightarrow{\texttt{I}}$	992	1 038	1 012	1 018	1 017	1 011	1 064
$\overrightarrow{\texttt{A+I}}$	1 117	**1 163**	1 121	1 150	1 138	1 136	1 178
$\overrightarrow{\texttt{M}_2\texttt{+I}}$	1 113	1 153	1 112	1 141	1 135	1 134	1 170
$\overrightarrow{\texttt{S}_{1k}\texttt{+I}}$	1 100	1 135	1 099	1 118	1 108	1 112	1 152
$\overrightarrow{\texttt{oracle}}$	1 130	1 185	1 136	1 159	1 152	1 150	1 204

Table 7
"Domain Dominance": the row x and column y shows the number of domains where the method x solved more tasks than the method y. "Task Dominance": the row x and column y shows the number of tasks solved by x but not by y. "tot": the overall number of solved tasks (coverage).

	Domain Dominance								Task Dominance								
	$\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$	$\overrightarrow{\texttt{M}_2\texttt{+I}}$-$\overleftarrow{\texttt{A+I}}$	$\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$	$\overrightarrow{\texttt{A+I}}$	$\updownarrow\texttt{b}$	$\uparrow\texttt{b}$	$\downarrow\texttt{I}$	$\downarrow\texttt{b}$	$\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$	$\overrightarrow{\texttt{M}_2\texttt{+I}}$-$\overleftarrow{\texttt{A+I}}$	$\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$	$\overrightarrow{\texttt{A+I}}$	$\updownarrow\texttt{b}$	$\uparrow\texttt{b}$	$\downarrow\texttt{I}$	$\downarrow\texttt{b}$	tot
$\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$	–	**16**	**21**	**15**	**26**	**37**	**41**	**37**	–	**31**	**53**	**62**	**128**	**230**	**350**	**377**	1 163
$\overrightarrow{\texttt{M}_2\texttt{+I}}$-$\overleftarrow{\texttt{A+I}}$	4	–	**12**	13	**23**	**35**	**39**	**36**	9	–	**37**	**60**	**120**	**220**	**326**	**363**	1 141
$\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$	3	9	–	10	**20**	**31**	**39**	**35**	11	17	–	**43**	**118**	**203**	**307**	**343**	1 121
$\overrightarrow{\texttt{A+I}}$	8	**15**	**15**	–	**22**	**31**	**39**	**37**	16	36	39	–	**128**	**184**	**320**	**351**	1 117
$\overrightarrow{\texttt{b}}$	6	7	11	10	–	**23**	**35**	**36**	20	34	52	66	–	**115**	**260**	**262**	1 055
$\overrightarrow{\texttt{b}}$	4	5	6	4	2	–	**26**	**30**	10	22	25	10	3	–	**186**	**190**	943
$\overleftarrow{\texttt{I}}$	0	1	2	1	7	12	–	**22**	5	3	4	21	23	61	–	**86**	818
$\overleftarrow{\texttt{b}}$	1	4	5	4	1	6	16	–	9	17	17	29	2	42	63	–	795

with operator-potential heuristics and blind symbolic search. The baseline of blind bi-directional symbolic search ($\overleftrightarrow{\texttt{b}}$), which was the state-of-the-art variant of symbolic search until now, solved 1 055 tasks, but all variants of GHSETA* using an operator-potential heuristic other than I in the forward direction overcome this result—even the forward-only GHSETA*. The table also shows that using operator-potential heuristics in the forward direction has much bigger impact on the overall number of solved tasks than operator-potential heuristics in the backward direction. This is in-line with our previous findings regarding forward-only and backward-only GHSETA*.

Table 7 provides even more insights as it compares selected bi-directional variants with operator-potential heuristics with the best-performing variants of the forward-only and backward-only GHSETA* with operator-potential heuristics and the blind variants of symbolic search in all directions. For the bi-directional GHSETA*, we selected the best-performing variant $\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$ and two more best-performing variants among those that do not use the blind backward search. The bi-directional variants with operator-potential heuristics are clearly superior to the forward-only and backward-only GHSETA* and blind variants in the overall number of solved tasks, in the number of domains where one variant solves more tasks than the other, and the number of tasks solved by one variant but not the other. The only exception is $\overrightarrow{\texttt{A+I}}$ that dominates $\overrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$ and $\overrightarrow{\texttt{M}_2\texttt{+I}}$-$\overleftarrow{\texttt{A+I}}$ in more domains than the other way around (although it is still worse in the overall numbers).



(a) Average number of BDD nodes per expanded BDD (i.e., average size of a BDD).
(b) Average runtime (in seconds) per expanded BDD.
(c) Number of expanded BDDs (i.e., number of expanded sets of states).
(d) Sum of expanded BDD nodes.
(e) Runtime in seconds.

Fig. 10. Per-task comparison between the best-performing variant of backward GHSETA* ($\overleftarrow{\texttt{I}}$) and the backward blind symbolic search ($\overleftarrow{\texttt{b}}$).

Closer inspection of the experimental results showed that there are only 29 and 10 tasks solved by $\overleftarrow{\texttt{b}}$ and $\overleftarrow{\texttt{I}}$, respectively, that are not solved by $\overrightarrow{\texttt{A+I}}$. This also at least partially explains why the best variant of the bi-directional symbolic search is $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$ and not $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$ even though the backward GHSETA* with I has higher overall coverage than the backward blind search: $\overleftarrow{\texttt{I}}$ seems to work better than $\overleftarrow{\texttt{b}}$ in tasks that can already be solved by $\overrightarrow{\texttt{A+I}}$. Thus, the blind backward search seems to be more complementary to the forward symbolic search with operator-potential heuristics.

Table 6 also shows the results for "oracles", i.e., for each individual task we select the best variant in the respective direction to see what is the potential of the proposed techniques. In other words, oracle shows the best possible result if we knew for each task in advance what is the best search technique from those introduced in this paper. The numbers indicate that if we better understand what makes one operator-potential heuristic better suited for a symbolic search in a particular task than the other, we may be able to further improve the performance of the proposed techniques. For example, comparing $\overleftrightarrow{\texttt{oracle}}$-$\overleftarrow{\texttt{b}}$ (1 185) and $\overleftrightarrow{\texttt{oracle}}$-$\overleftarrow{\texttt{oracle}}$ (1 204), one can observe that there are 19 instances that are only solved when operator-potential heuristics are used within the backward search.

As in the previous cases, we also analyze in more detail the effect of the proposed methods on the number and sizes of BDDs representing sets of states and the runtime. Fig. 10 compares the best-performing bi-directional variant $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$ with (on the top row) the baseline $\overleftrightarrow{\texttt{b}}$ (which was up until now the best symbolic planner), and (on the bottom row) with $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$ to see how using a different operator-potential heuristics in the backward direction affects the search. Recall that $\overleftarrow{\texttt{I}}$ performs better than $\overleftarrow{\texttt{b}}$, but $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$ performs better than $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$—there are only 11 tasks solved by $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$ that are not solved by $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$, but 53 tasks solved by $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$ and not by $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$ (cf. Table 7).

The comparison to $\overleftrightarrow{\texttt{b}}$ seems to replicate the improvements that we observed when comparing $\overrightarrow{\texttt{A+I}}$ and $\overrightarrow{\texttt{b}}$ (cf. Fig. 7). This is not surprising as the difference between the methods is exactly replacing blind forward search with $\overrightarrow{\texttt{A+I}}$, which was shown to greatly improve performance in the unidirectional search case. The bottom row of Fig. 10 comparing $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$ and $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$ shows that replacing b with I in the backward direction of the bi-directional search has much less profound effect than in the backward-only search (cf. Fig. 9). Nevertheless, we can still observe that using operator-potential heuristics in the backward direction (instead of blind search) results in more expanded BDDs (sets of states) because I induces more fine-grained partitioning of sets of states (Fig. 10c, bottom), and we can see that the size of the expanded BDDs tends to be smaller with I (Fig. 10a, bottom). However, as we already noted, $\overrightarrow{\texttt{A+I}}$ seems to be more complementary with $\overleftarrow{\texttt{b}}$ than with $\overleftarrow{\texttt{I}}$ which results in a better overall performance of $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$.

5.6. Comparison to state-of-the-art

What remains is the comparison to other state-of-the-art planning methods. From the newly proposed methods, we consider the best variants of GHSETA* in all directions ($\overrightarrow{\texttt{A+I}}$, $\overleftarrow{\texttt{I}}$, $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$), and the bi-directional GHSETA* combining the best forward and backward variant ($\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$). We compare those to heuristic state-based planners (lmc, ms, comp2, and scrp), the blind bi-directional symbolic search ($\overleftrightarrow{\texttt{b}}$), and the symbolic search with pattern databases (cgm). Table 8 compares the methods by counting the number of domains where one method solved more tasks than the other, and the number of tasks solved by one method but not the other.

$\overrightarrow{\texttt{A+I}}$, $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$, and $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$ perform better than any other compared method in the overall number of solved tasks. Since cgm performs worse than the blind bi-directional symbolic search, it is not surprising it performs much worse than our best methods. Moreover, there is not much complementarity between cgm and our best methods, i.e., there are only few domains or tasks where cgm performs better than $\overrightarrow{\texttt{A+I}}$, $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{b}}$, or $\overleftrightarrow{\texttt{A+I}}$-$\overleftarrow{\texttt{I}}$. A similar picture can be observed with the heuristic planners lmc and ms: They perform significantly worse than our best methods and there is not much complementarity.

Table 8
"Domain Dominance": the row x and column y shows the number of domains where the method x solved more tasks than the method y. "Task Dominance": the row x and column y shows the number of tasks solved by x but not by y. "tot": the overall number of solved tasks (coverage). For cgm, we considered only the subset of domains supported by the planner, i.e., both the row and columns for cgm disregard domains caldera, cavediving, GED, maintenance, movie, mprime, snake, spider, termes, and trucks.

	Domain Dominance										Task Dominance										
	$\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$	$\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{I}}$	$\overrightarrow{\text{A+I}}$	scrp	comp2	$\leftrightarrow$b	lmc	ms	$\overleftarrow{\text{I}}$	cgm	$\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$	$\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{I}}$	$\overrightarrow{\text{A+I}}$	scrp	comp2	$\leftrightarrow$b	lmc	ms	$\overleftarrow{\text{I}}$	cgm	tot
$\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$	–	**21**	**15**	17	**23**	**26**	**36**	**39**	**41**	**36**	–	**53**	**62**	**137**	**117**	**128**	**277**	**321**	**350**	**481**	1 163
$\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{I}}$	3	–	10	14	17	**20**	**34**	**35**	**39**	**34**	11	–	**43**	**120**	**106**	**118**	**246**	**288**	**307**	**451**	1 121
$\overrightarrow{\text{A+I}}$	8	**15**	–	15	**20**	**22**	**34**	**38**	**39**	**35**	16	39	–	**113**	**115**	**128**	**254**	**271**	**320**	**459**	1 117
scrp	**19**	**25**	**25**	–	**19**	**27**	**36**	**37**	**35**	**29**	77	102	99	–	**119**	**179**	**230**	**251**	**362**	**463**	1 103
comp2	10	**18**	16	16	–	**21**	**33**	**38**	**38**	**34**	50	81	94	112	–	**86**	**218**	**245**	**303**	**424**	1 096
$\leftrightarrow$b	6	11	10	14	12	–	**25**	**32**	**35**	**34**	20	52	66	131	45	–	**188**	**232**	**260**	**389**	1 055
lmc	1	5	5	2	6	9	–	**21**	**28**	**28**	15	26	38	28	23	34	–	**100**	**184**	**299**	901
ms	1	3	1	1	4	9	15	–	**21**	**28**	17	26	13	7	8	36	58	–	**160**	**274**	859
$\overleftarrow{\text{I}}$	0	2	1	7	5	7	16	16	–	**27**	5	4	21	77	25	23	101	119	–	**255**	818
cgm	1	2	2	6	0	1	8	8	10	–	5	10	16	46	5	3	51	62	62	–	518

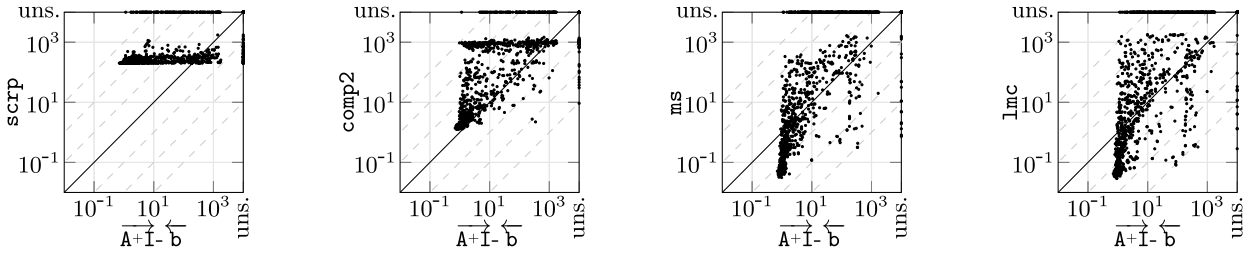


Fig. 11. Per-task comparison of the runtime (in seconds) of the best variant of GHSETA* against the explicit state search methods scrp and comp2.

scrp and comp2 both solve less tasks than our best methods overall, but they also seem to be complementary to our approach. In particular, the number of domains where scrp performs better than our methods is higher than the other way around. In fact, scrp is the best-performing planner (from the ones compared here) in 28 domains, whereas $\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$ is the best-performing planner in 22 domains ($\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{I}}$ in 14 domains, and $\overrightarrow{\text{A+I}}$ in 15 domains). Nevertheless, the overall numbers are in favor of the bi-directional GHSETA* with operator-potential heuristics, and the difference seems to be spread over large number of structurally different domains.

In terms of runtime, Fig. 11 shows a comparison of our best approach, $\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$, against several explicit-state search planners. The comparison with scrp and comp2 is not very insightful as they use an expensive preprocessing phase to compute the heuristic with a fixed time limit of 900 and 300 seconds, respectively. Due to this, $\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$ is faster in the majority of instances. The comparison with lmc and ms shows that, despite the huge advantage in coverage of $\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$, there are still a number of instances solved faster by these planners. This is partially due to the preprocessing phase of $\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$, which requires some time to compute the potential heuristics and initialize the data-structures to perform symbolic search. But overall, $\overrightarrow{\text{A+I}}$-$\overleftarrow{\text{b}}$ is still up to several orders of magnitude faster on many instances.

6. Conclusion

Heuristic state space search and symbolic search are complementary enhancements to the same basic algorithm—state space search—through the use of heuristic search guidance functions h, and of compact state-set representations, respectively. It is natural to combine both approaches, yet that combination has not been an unqualified success. One key reason for this is that, in symbolic search, h must be (efficiently) applicable to sets of states rather than individual ones. Here we show that potential heuristics can be reformulated in a manner allowing to do just that. The resulting methods empirically do not tend to suffer from the second key problem (detrimental state partitionings). They soundly beat the previous state of the art in symbolic search for optimal planning; they are on par with, as well as highly complementary to, the state of the art in optimal heuristic search planning.

This result boosts our ability to plan optimally, and it re-emphasizes the role of symbolic search, in particular heuristic symbolic search, as part of the state of the art, suggesting that further research effort may be well placed in this area which recently received scant attention.

A specific question opened up by our research is whether the key to our method—the transformation of heuristic values into a sum of heuristic-value changes per operator—may be applicable to other kinds of heuristic functions as well. For example, for abstraction heuristics this is certainly not true per se, as the change in heuristic value (abstract goal distance) is highly dependent on the state in general. But perhaps abstractions can be designed so as to avoid that phenomenon. Similarly, it may be possible to adjust the design of other admissible estimators, like landmark heuristics, to this end.

Beyond this, there is a number of issues that our work sheds light upon, and that would be worth being explored more broadly. Our experimental analysis shows that potential heuristics defined in previous work obtain great performance in forward search, and a mild improvement in backward search. This suggests to investigate what kind of potential heuristics can do better in each direction, as well as defining new optimization criteria for potential heuristics, and/or investigating whether higher-dimensional potential heuristics [39] can further improve performance. More generally, a key issue is to improve our understanding of what makes a heuristic good in symbolic search. Operator-potential heuristics offer a clear positive example, in contrast to previous analysis [51]. A promising avenue of research is to characterize what kind of heuristics induce good state partitionings for sets of states represented as BDDs, and how the choice of representation (e.g., using EVMDDs instead [8,35,50]) affects the usefulness of different heuristic functions. Another promising line of research is on how to best make use of heuristics in symbolic bi-directional search. In recent years, there have been significant advances in explicit state bi-directional heuristic search [1,3,31,46,47], which could shed light on how to make the most out of heuristics in the symbolic search case too.

CRediT authorship contribution statement

Daniel Fišer: Writing – review & editing, Writing – original draft. **Álvaro Torralba:** Writing – review & editing, Writing – original draft. **Jörg Hoffmann:** Writing – review & editing, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All data and source code are publicly available via repositories referenced in the paper.

Acknowledgements

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—project number 389792660—TRR 248 (see https://perspicuous-computing.science).

Appendix A. General case: path-dependent heuristics

In this section, we consider the general case of path-dependent heuristics, where we allow for the heuristic value of a state s to depend on the path used to reach s from the initial state (in forward search), or the goal (in backward search). This opens several new possibilities:

1. We can lose the assumption that $\mathrm{vars}(\mathrm{pre}(o)) = \mathrm{vars}(\mathrm{eff}(o))$ for every operator o, i.e., we move from the normalized FDR to the FDR in its general form. So, for the rest this section, let $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G \rangle$ denote a planning task (not necessarily normalized).
2. We no longer require that the operator-potential is the exact change in heuristic value. Rather, it suffices if this is an admissible approximation. This allows to infer potential functions using the LP encoding and rounding potentials down to the nearest integer instead of using the MIP encoding.

However, we observe in the experiments (see results below) that normalizing the task to have state-dependent heuristics is almost always superior in practice. As the general case complicates the theory and proofs, we decided to focus the main part of the paper on the conceptually simplest (and in practice better performing) case and only consider path-dependent heuristics in this appendix.

The definitions of potential functions and potential heuristics (Definition 1) do not change for the general case, but the sufficient conditions under which the potential heuristic is consistent (Equation (3)) need to be adapted because it can happen that there is an operator o that affects a variable for which a precondition is not defined, i.e., $\mathrm{vars}(\mathrm{eff}(o)) \setminus \mathrm{vars}(\mathrm{pre}(o)) \neq \emptyset$. As in the previous case, we borrow the formulation from Fišer et al. [20]. Recall that $D_o(V)$ denotes a disambiguation of V for $\mathrm{pre}(o) \cup \mathrm{prv}(o)$ for every $o \in \mathcal{O}$. Here, we assume $D_o(V)$ is given to us for every $o \in \mathcal{O}$ and every $V \in \mathcal{V}$ (e.g., it has been computed by using the method by Fišer et al. [20]).

Theorem 15. *Let P denote a potential function. If Equation (2) holds and for every operator $o \in \mathcal{O}$ it holds that*

$$\sum_{V \in \mathrm{vars}(\mathrm{eff}(o))} \max_{f \in D_o(V)} \mathrm{P}(f) - \sum_{f \in \mathrm{eff}(o)} \mathrm{P}(f) \leq \mathrm{cost}(o), \tag{A.1}$$

then $h^{\mathrm{P}}_{\mathrm{fw}}$ is goal-aware, forward consistent, and forward admissible.

We show that for non-normalized FDR tasks, heuristics computed as sums of operator-potential values over sequences of operators lead to path-dependent heuristics. Similarly to a forward heuristic h_{fw}, a **path-dependent forward heuristic** $\widetilde{h}_{\mathrm{fw}} : \mathcal{E}_{\mathrm{fw}} \mapsto \mathbb{R} \cup \{\infty\}$

estimates the cost of optimal s-plans, but it takes sequences of operators applicable in I instead of reachable states as its inputs, i.e., two different sequences of operators $\pi, \pi' \in \mathcal{E}_{\text{fw}}$ such that $\pi[\![I]\!] = \pi'[\![I]\!] = s \in \mathcal{S}_{\text{fw}}$ can result in two different heuristic values $\widetilde{h}_{\text{fw}}(\pi) \neq \widetilde{h}_{\text{fw}}(\pi')$. A path-dependent forward heuristic $\widetilde{h}_{\text{fw}}$ is called **forward admissible** if $\widetilde{h}_{\text{fw}}(\pi) \leq h^{\star}_{\text{fw}}(s)$ for every reachable state $s \in \mathcal{S}_{\text{fw}}$ and every sequence of operators $\pi \in \mathcal{E}_{\text{fw}}$ such that $\pi[\![I]\!] = s$.

A **path-dependent backward heuristic** $\widetilde{h}_{\text{bw}} : \mathcal{E}_{\text{bw}} \mapsto \mathbb{R} \cup \{\infty\}$ is defined analogously to the path-dependent forward heuristic, i.e., it takes s-plans and estimates the cost of the optimal I-s-paths. A path-dependent backward heuristic $\widetilde{h}_{\text{bw}}$ is called **backward admissible** if $\widetilde{h}_{\text{bw}}(\pi) \leq h^{\star}_{\text{bw}}(s)$ for every $s \in \mathcal{S}_{\text{bw}}$ and every $\pi \in \mathcal{E}_{\text{bw}}$ such that π is applicable in s and $\pi[\![s]\!]$ is a goal state.

A.1. General operator-potential function

The previous definition of the operator-potential function (Definition 3) was pertinent to the normalized FDR tasks only. Here, we need to generalize the definition to cover also cases where operators affect variables not defined in their preconditions. Note that similarly to Definition 3, the general operator-potential function $\widetilde{\mathtt{Q}}$ is defined using the left hand side of the consistency condition on potential function (Equation (A.1)) with the opposite sign.

Definition 16. Given a potential function $\mathtt{P}$, a function $\widetilde{\mathtt{Q}} : \mathcal{O} \mapsto \mathbb{R}$ is called a **general operator-potential function** for $\mathtt{P}$ if

$$\widetilde{\mathtt{Q}}(o) \leq \sum_{f \in \mathtt{eff}(o)} \mathtt{P}(f) - \sum_{V \in \mathtt{vars}(\mathtt{eff}(o))} \max_{f \in D_o(V)} \mathtt{P}(f) \tag{A.2}$$

for every operator $o \in \mathcal{O}$.

Typically, we would like to use the largest value satisfying the inequality, i.e.,

$$\widetilde{\mathtt{Q}}(o) = \sum_{f \in \mathtt{eff}(o)} \mathtt{P}(f) - \sum_{V \in \mathtt{vars}(\mathtt{eff}(o))} \max_{f \in D_o(V)} \mathtt{P}(f), \tag{A.3}$$

as that leads to the most informative heuristic. However, it is still safe to use lower values if that is convenient for some reason. For example, this allows us to compute potential functions with floating-point operator potentials and round them down to the nearest integer. Also note that if the input planning task is normalized, then $\widetilde{\mathtt{Q}}$ computed with Equation (A.3) is equal to $\mathtt{Q}$, because in normalized tasks, $D_o(V)$ is a singleton for every $V \in \mathtt{vars}(\mathtt{eff}(o))$, i.e., $\bigcup_{V \in \mathtt{vars}(\mathtt{eff}(o))} D_o(V) = \mathtt{pre}(o)$.

Next, we show that $\widetilde{\mathtt{Q}}(o)$ (constructed from the potential function $\mathtt{P}$) is a lower bound on the change of the heuristic value of $h^{\mathtt{P}}_{\text{fw}}$ induced by the operator o.

Proposition 17. *Let $\widetilde{\mathtt{Q}}$ denote a general operator-potential function for $\mathtt{P}$, $s \in \mathcal{S}_{\text{fw}}$ denote a forward reachable state, and let $o \in \mathcal{O}$ denote an operator applicable in s. Then $\sum_{f \in s} \mathtt{P}(f) + \widetilde{\mathtt{Q}}(o) \leq \sum_{f \in o[\![s]\!]} \mathtt{P}(f)$.*

Proof. We prove the case where $\widetilde{\mathtt{Q}}(o) = \sum_{f \in \mathtt{eff}(o)} \mathtt{P}(f) - \sum_{V \in \mathtt{vars}(\mathtt{eff}(o))} \max_{f \in D_o(V)} \mathtt{P}(f)$, the general case from Definition 16 easily follows. Let $t = o[\![s]\!] \setminus \mathtt{eff}(o)$ denote the set of facts unaffected by the operator o, and let $x = s \setminus t$ denote the set of facts changed by o. So, we have $\sum_{f \in s} \mathtt{P}(f) = \sum_{f \in t} \mathtt{P}(f) + \sum_{f \in x} \mathtt{P}(f)$ and $\sum_{f \in o[\![s]\!]} \mathtt{P}(f) = \sum_{f \in t} \mathtt{P}(f) + \sum_{f \in \mathtt{eff}(o)} \mathtt{P}(f)$. Therefore, we need to prove that $\sum_{f \in x} \mathtt{P}(f) + \sum_{f \in \mathtt{eff}(o)} \mathtt{P}(f) - \sum_{V \in \mathtt{vars}(\mathtt{eff}(o))} \max_{f \in D_o(V)} \mathtt{P}(f) \leq \sum_{f \in \mathtt{eff}(o)} \mathtt{P}(f)$ which holds iff $\sum_{f \in x} \mathtt{P}(f) - \sum_{V \in \mathtt{vars}(\mathtt{eff}(o))} \max_{f \in D_o(V)} \mathtt{P}(f) \leq 0$ holds. It is easy to see that $\mathtt{vars}(x) = \mathtt{vars}(\mathtt{eff}(o))$ and from the definition of disambiguation it follows that for every $V \in \mathtt{vars}(x)$ it holds that $x[V] \in D_o(V)$. Therefore, for every $V \in \mathtt{vars}(x)$, we have that $\mathtt{P}(x[V]) \leq \max_{f \in D_o(V)} \mathtt{P}(f)$, which concludes the proof. □

Unlike $\mathtt{Q}$ in case of normalized FDR tasks, $\widetilde{\mathtt{Q}}$ for non-normalized tasks does not necessarily capture the change of the heuristic value exactly, even if we take the maximal value satisfying the inequality in Equation (A.2). Consider the simple non-normalized planning task in Fig. A.1 and states "aX" and "bX", and the operator o_1. The $h^{\mathtt{P}}_{\text{fw}}$-value for "aX" is zero and $h^{\mathtt{P}}_{\text{fw}}$-value for "bX" is one, i.e., the change of the heuristic value is one, but $\widetilde{\mathtt{Q}}(o_1) = 0$ (as Equation (A.2) needs to account also for the self-loop resulting from applying o_1 in the state "bX").

A.2. Forward direction

Path-dependent operator-potential forward heuristics are defined analogously to their state-dependent counterparts (Definition 6), except that here we use $\widetilde{\mathtt{Q}}$ instead of $\mathtt{Q}$.

Definition 18. Let $\widetilde{\mathtt{Q}}$ denote a general operator-potential function for $\mathtt{P}$. A **path-dependent operator-potential forward heuristic** $\widetilde{h}^{\mathtt{Q}}_{\text{fw}} : \mathcal{E}_{\text{fw}} \mapsto \mathbb{R} \cup \{\infty\}$ for $\widetilde{\mathtt{Q}}$ is defined as

$$\widetilde{h}^{\mathtt{Q}}_{\text{fw}}(\pi) = \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \widetilde{\mathtt{Q}}(o_i) \tag{A.4}$$

$\mathcal{V} = \{v_1, v_2\}$, $\mathrm{dom}(v_1) = \{a, b\}$, $\mathrm{dom}(v_2) = \{X, Y\}$

$I = \{\langle v_1, a\rangle, \langle v_2, X\rangle\}$, $G = \{\langle v_1, b\rangle, \langle v_2, Y\rangle\}$

$o \in \mathcal{O}$	prv(o)	pre(o)	eff(o)	cost(o)	$\widetilde{\mathtt{Q}}(o)$
o_1	$\langle v_2, X\rangle$	$\emptyset$	$\langle v_1, b\rangle$	1	0
o_2	$\emptyset$	$\emptyset$	$\langle v_2, Y\rangle$	1	-1
o_3	$\langle v_2, Y\rangle$	$\langle v_1, a\rangle$	$\langle v_1, b\rangle$	1	1

f	P(f)
$\langle v_1, a\rangle$	0
$\langle v_1, b\rangle$	1
$\langle v_2, X\rangle$	0
$\langle v_2, Y\rangle$	-1

state s	$h^{\mathtt{P}}_{\mathrm{fw}}(s)$
$\{\langle v_1, a\rangle, \langle v_2, X\rangle\}$	0
$\{\langle v_1, a\rangle, \langle v_2, Y\rangle\}$	-1
$\{\langle v_1, b\rangle, \langle v_2, X\rangle\}$	1
$\{\langle v_1, b\rangle, \langle v_2, Y\rangle\}$	0

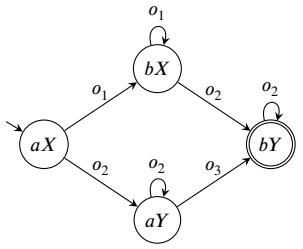


Fig. A.1. Example planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, I, G\rangle$ showing path dependency of operator-potential heuristics for non-normalized FDR tasks. $\widetilde{\mathtt{Q}}$ is computed using Equation (A.3).

for any sequence of operators $\pi = \langle o_1, \ldots, o_n\rangle$ applicable in I.

Observe that $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ can, indeed, be path-dependent for non-normalized planning tasks. Consider the example planning task in Fig. A.1 again. The goal state "bY" can be reached from the initial state "aX" by two different paths, $\pi = \langle o_1, o_2\rangle$ and $\pi' = \langle o_2, o_3\rangle$, and we obtain two different heuristic values for π and π'. Namely, $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}(\pi) = h^{\mathtt{P}}_{\mathrm{fw}}(I) + \widetilde{\mathtt{Q}}(o_1) + \widetilde{\mathtt{Q}}(o_2) = -1$ and $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}(\pi') = h^{\mathtt{P}}_{\mathrm{fw}}(I) + \widetilde{\mathtt{Q}}(o_2) + \widetilde{\mathtt{Q}}(o_3) = 0$.

Next, we show that if the underlying potential heuristic $h^{\mathtt{P}}_{\mathrm{fw}}$ is forward admissible, then the path-dependent operator-potential forward heuristic $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ is also forward admissible. This follows simply from the fact that $\widetilde{\mathtt{Q}}$ provides lower bounds on the change of the heuristic value of $h^{\mathtt{P}}_{\mathrm{fw}}$ induced by each operator. So, if we start from the admissible estimate for the initial state, then adding these lower bounds results in an admissible estimate for all forward reachable states.

Theorem 19. *If $h^{\mathtt{P}}_{\mathrm{fw}}$ is forward admissible, then $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ is forward admissible.*

Proof. We need to prove that for any $\pi = \langle o_1, \ldots, o_n\rangle \in \mathcal{E}_{\mathrm{fw}}$ and $s = \pi[\![I]\!]$ it holds that $\sum_{f\in I} \mathtt{P}(f) + \sum_{i\in[n]} \widetilde{\mathtt{Q}}(o_i) \leq \sum_{f'\in s} \mathtt{P}(f')$, which we will prove by induction.

The claim clearly holds for the empty sequence π because $h^{\mathtt{P}}_{\mathrm{fw}}(I) = \widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}(I)$. So, we assume $\sum_{f\in I} \mathtt{P}(f) + \sum_{i\in[k]} \widetilde{\mathtt{Q}}(o_i) \leq \sum_{f\in s'} \mathtt{P}(f)$ holds for some $0 < k < n$, $\pi' = \langle o_1, \ldots, o_k\rangle$, and $s' = \pi'[\![I]\!]$, and we need to prove that $\sum_{f\in I} \mathtt{P}(f) + \sum_{i\in[k+1]} \widetilde{\mathtt{Q}}(o_i) \leq \sum_{f\in o_{k+1}[\![s']\!]} \mathtt{P}(f)$ also holds.

From the assumption $\sum_{f\in I} \mathtt{P}(f) + \sum_{i\in[k]} \widetilde{\mathtt{Q}}(o_i) \leq \sum_{f\in s'} \mathtt{P}(f)$ it follows that

$$\sum_{f\in I} \mathtt{P}(f) + \sum_{i\in[k]} \widetilde{\mathtt{Q}}(o_i) + \widetilde{\mathtt{Q}}(o_{k+1}) \leq \sum_{f\in s'} \mathtt{P}(f) + \widetilde{\mathtt{Q}}(o_{k+1}).$$

From Proposition 17, it follows that $\sum_{f\in s'} \mathtt{P}(f) + \widetilde{\mathtt{Q}}(o_{k+1}) \leq \sum_{f\in o_{k+1}[\![s']\!]} \mathtt{P}(f)$. Therefore, we have

$$\sum_{f\in I} \mathtt{P}(f) + \sum_{i\in[k+1]} \widetilde{\mathtt{Q}}(o_i) = \sum_{f\in I} \mathtt{P}(f) + \sum_{i\in[k]} \widetilde{\mathtt{Q}}(o_i) + \widetilde{\mathtt{Q}}(o_{k+1}) \leq \sum_{f\in s'} \mathtt{P}(f) + \widetilde{\mathtt{Q}}(o_{k+1}) \leq \sum_{f\in o_{k+1}[\![s']\!]} \mathtt{P}(f),$$

which concludes the proof. □

This allows us to use operator-potential heuristics also for non-normalized planning tasks, but, as we noted before, it almost always pays off to normalize planning tasks and use $h^{\mathtt{Q}}_{\mathrm{fw}}$ (which is forward consistent) instead of using the path-dependent variant $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ with the original planning task.

A.3. Backward direction

Using $\widetilde{\mathtt{Q}}$ instead of $\mathtt{Q}$ to compute path-dependent operator-potential backward heuristics allows us to use it also for non-normalized planning tasks and without goal-splitting. Note that in the case of path-dependent heuristics we also require Equation (A.1) to hold for the potential heuristic $\mathtt{P}$, which is necessary to ensure backward admissibility of the heuristics.

Definition 20. Let $\widetilde{\mathtt{Q}}$ denote a general operator-potential function for $\mathtt{P}$ such that Equation (2) and Equation (A.1) hold. A **path-dependent operator-potential backward heuristic** $\widetilde{h}^{\mathtt{Q}}_{\mathrm{bw}} : \mathcal{E}_{\mathrm{bw}} \mapsto \mathbb{R} \cup \{\infty\}$ for $\mathtt{Q}$ is defined as

Table A.1
Comparison of the path-dependent operator-potential forward heuristics ($\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$) on the original planning task, and the (consistent) operator-potential forward heuristics ($h^{\mathtt{Q}}_{\mathrm{fw}}$) on the normalized tasks. Whenever the original planning task is already normalized, $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ is used as a consistent heuristic, i.e., it equals to $h^{\mathtt{Q}}_{\mathrm{fw}}$. We compare the number of domains where one method solved more tasks than the other, the number of tasks solved by one method but not the other, and the overall number of solved tasks.

		I	A+I	S_{1k}+I	M_2+I
$\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$	#domains with higher coverage than $h^{\mathtt{Q}}_{\mathrm{fw}}$	1	1	0	1
	#tasks solved by $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ but not by $h^{\mathtt{Q}}_{\mathrm{fw}}$	3	1	1	2
	overall number of solved tasks	952	1 081	1 057	1 078
$h^{\mathtt{Q}}_{\mathrm{fw}}$	#domains with higher coverage than $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$	11	11	11	10
	#tasks solved by $h^{\mathtt{Q}}_{\mathrm{fw}}$ but not by $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$	43	37	44	37
	overall number of solved tasks	992	1 117	1 100	1 113

$$\widetilde{h}^{\mathtt{Q}}_{\mathrm{bw}}(\pi) = \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \widetilde{\mathtt{Q}}(o_i) \tag{A.5}$$

for every sequence of operators $\pi = \langle o_1, \dots, o_n \rangle \in \mathcal{E}_{\mathrm{bw}}$, i.e., for every s-plan π.

Theorem 21. *$\widetilde{h}^{\mathtt{Q}}_{\mathrm{bw}}$ is backward admissible.*

Proof. Any estimate for a backward dead-end state is admissible. Let $\pi = \langle o_1, \dots, o_n \rangle$ denote a plan, and let $m \in [n]$. It is enough to show that $\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [m+1,n]} \widetilde{\mathtt{Q}}(o_i) \leq \sum_{j \in [m]} \mathrm{cost}(o_j)$. We show this for $\widetilde{\mathtt{Q}}(o) = \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) - \sum_{V \in \mathrm{vars}(\mathrm{eff}(o))} \max_{f \in D_o(V)} \mathtt{P}(f)$, and it is clear that the inequality holds for lower values of $\widetilde{\mathtt{Q}}(o)$.

Since Equation (2) and Equation (A.1) hold, it follows from Theorem 15 that $h^{\mathtt{P}}_{\mathrm{fw}}$ is forward consistent, goal-aware, and forward admissible, and therefore $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ is forward admissible Theorem 19. Therefore, we have that

$$\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}(\pi[\![I]\!]) = \sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [n]} \widetilde{\mathtt{Q}}(o_i) = \sum_{f \in I} \mathtt{P}(f) + \sum_{j \in [m]} \widetilde{\mathtt{Q}}(o_j) + \sum_{i \in [m+1,n]} \widetilde{\mathtt{Q}}(o_i) \leq 0.$$

Therefore it follows that $\sum_{f \in I} \mathtt{P}(f) + \sum_{i \in [m+1,n]} \widetilde{\mathtt{Q}}(o_i) \leq -\sum_{j \in [m]} \widetilde{\mathtt{Q}}(o_j)$.

From Definition 16 and Equation (A.1), it follows that

$$\widetilde{\mathtt{Q}}(o) = \sum_{f \in \mathrm{eff}(o)} \mathtt{P}(f) - \sum_{V \in \mathrm{vars}(\mathrm{eff}(o))} \max_{f \in D_o(V)} \mathtt{P}(f) \geq -\mathrm{cost}(o),$$

therefore $-\widetilde{\mathtt{Q}}(o) \leq \mathrm{cost}(o)$ for every operator o, therefore it holds that $-\sum_{j \in [m]} \widetilde{\mathtt{Q}}(o_j) \leq \sum_{j \in [m]} \mathrm{cost}(o_j)$, which concludes the proof. □

A.4. Symbolic search with path-dependent heuristics

Algorithm 1 describes the GHSETA* algorithm when assuming consistent heuristics. However, path-dependent heuristics may create inconsistencies, leading to states being expanded with sub-optimal g-values. This, in turn, may lead to states being incorrectly pruned in lines 8 or 13, and the algorithm could return a sub-optimal plan.

However, the algorithm can easily be adapted to support inconsistent (and path-dependent) heuristics by re-expanding any state if it is reached again with a lower g-value. This requires to replace the `closed` set (which is used in Algorithm 1 to hold all expanded states) by multiple subsets, $\mathtt{closed}_g$, each of which contains the set of all states expanded with the corresponding g-value. Therefore, in line 11, states are inserted in $\mathtt{closed}_g$. And in lines 8 and 13, we loop over all $g' \leq g$ to remove any state in $\mathtt{closed}_{g'}$ and $\mathtt{closed}_{g'+c}$, respectively.

A.5. Results

We evaluate symbolic search with state-dependent ($h^{\mathtt{Q}}_{\mathrm{fw}}/h^{\mathtt{Q}}_{\mathrm{bw}}$) and path-dependent ($\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}/\widetilde{h}^{\mathtt{Q}}_{\mathrm{bw}}$) operator-potential heuristics. The state-dependent variant is the one we analyzed in detail in Section 5: It requires to normalize planning tasks and partitioning of goal states (in case of backward symbolic search). For path-dependent heuristics, we avoid normalizing the task and partitioning of goal states. But, in exchange, we need to re-expand states within the GHSETA* algorithm as explained above.

Table A.1 summarizes the comparison between the two variants in forward search. There are at most three tasks where using $\widetilde{h}^{\mathtt{Q}}_{\mathrm{fw}}$ is beneficial to increase coverage, whereas $h^{\mathtt{Q}}_{\mathrm{fw}}$ performs better in 37 to 44 tasks, depending on the optimization criteria of the underlying potential heuristics.

Table A.2 shows results for backward search. Here, we include as well results for the path-dependent heuristic on the normalized task (but without goal-splitting), to better understand the effect that task normalization and goal-splitting has on the overall performance.

Table A.2
Comparison of the path-dependent operator-potential backward heuristics on the original planning tasks ($\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig), on the tasks normalized with the "multiplication" method but without goal splitting ($\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm), and the operator-potential backward heuristics on normalized tasks with goal-splitting ($h^{\mathrm{Q}}_{\mathrm{bw}}$). We compare the number of domains where one method solved more tasks than the other, the number of tasks solved by one method but not the other, and the overall number of solved tasks.

		I	A+I	S_{1k}+I	M_2+I
$\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	#domains with higher coverage than $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	3	1	2	3
	#tasks solved by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig but not by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	3	3	3	4
	#domains with higher coverage than $h^{\mathrm{Q}}_{\mathrm{bw}}$	5	3	6	3
	#tasks solved by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig but not by $h^{\mathrm{Q}}_{\mathrm{bw}}$	14	12	19	13
	overall number of solved tasks	715	725	713	725
$\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	#domains with higher coverage than $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	5	5	6	3
	#tasks solved by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm but not by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	11	13	11	10
	#domains with higher coverage than $h^{\mathrm{Q}}_{\mathrm{bw}}$	5	2	8	3
	#tasks solved by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm but not by $h^{\mathrm{Q}}_{\mathrm{bw}}$	13	9	22	13
	overall number of solved tasks	723	735	721	731
$h^{\mathrm{Q}}_{\mathrm{bw}}$	#domains with higher coverage than $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	28	20	22	18
	#tasks solved by $h^{\mathrm{Q}}_{\mathrm{bw}}$ but not by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	117	81	76	81
	#domains with higher coverage than $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	25	19	21	21
	#tasks solved by $h^{\mathrm{Q}}_{\mathrm{bw}}$ but not by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	108	68	71	75
	overall number of solved tasks	818	794	770	793

Table A.3
Same as Table A.1 and Table A.2, but for bi-directional symbolic search. Note that in the case of $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$ -norm, we use inconsistent $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$, but forward consistent $h^{\mathrm{Q}}_{\mathrm{fw}}$ as the tasks are already normalized.

		$\overrightarrow{\mathrm{I}}$-$\overleftarrow{\mathrm{A}^+\mathrm{I}}$	$\overrightarrow{\mathrm{I}}$-$\overleftarrow{\mathrm{M}_2^+\mathrm{I}}$	$\overrightarrow{\mathrm{A}^+\mathrm{I}}$-$\overleftarrow{\mathrm{A}^+\mathrm{I}}$	$\overrightarrow{\mathrm{A}^+\mathrm{I}}$-$\overleftarrow{\mathrm{M}_2^+\mathrm{I}}$	$\overrightarrow{\mathrm{M}_2^+\mathrm{I}}$-$\overleftarrow{\mathrm{I}}$	$\overrightarrow{\mathrm{S}_{1k}^+\mathrm{I}}$-$\overleftarrow{\mathrm{S}_{1k}^+\mathrm{I}}$
$\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	#domains with higher coverage than $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	3	1	3	1	4	2
	#tasks solved by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig but not by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	8	5	5	3	5	5
	#domains with higher coverage than $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $h^{\mathrm{Q}}_{\mathrm{bw}}$	5	6	6	8	9	4
	#tasks solved by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig but not by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $h^{\mathrm{Q}}_{\mathrm{bw}}$	10	13	8	19	18	9
	overall number of solved tasks	987	985	1 104	1 104	1 083	1 080
$h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	#domains with higher coverage than $\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	14	14	14	15	14	8
	#tasks solved by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm but not by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	28	29	30	36	33	27
	#domains with higher coverage than $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $h^{\mathrm{Q}}_{\mathrm{bw}}$	5	5	6	8	8	5
	#tasks solved by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm but not by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $h^{\mathrm{Q}}_{\mathrm{bw}}$	8	12	11	22	22	10
	overall number of solved tasks	1 007	1 009	1 129	1 137	1 111	1 102
$h^{\mathrm{Q}}_{\mathrm{fw}}$ - $h^{\mathrm{Q}}_{\mathrm{bw}}$	#domains with higher coverage than $\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	14	16	17	18	14	11
	#tasks solved by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $h^{\mathrm{Q}}_{\mathrm{bw}}$ but not by $\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-orig	41	45	54	53	47	41
	#domains with higher coverage than $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	7	8	9	7	8	7
	#tasks solved by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $h^{\mathrm{Q}}_{\mathrm{bw}}$ but not by $h^{\mathrm{Q}}_{\mathrm{fw}}$ - $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$-norm	19	20	32	23	23	20
	overall number of solved tasks	1 018	1 017	1 150	1 138	1 112	1 112

As in the previous case, using path-dependent variant of operator-potential backward heuristics is rarely beneficial over using the state-dependent variant with goal-splitting $h^{\mathrm{Q}}_{\mathrm{bw}}$. Normalizing the task is actually an advantage, due to improving the informativeness of the operator-potential heuristics (as the max expression in Equation (A.2) is basically an admissible approximation).

Finally, as for $h^{\mathrm{Q}}_{\mathrm{fw}}$ and $h^{\mathrm{Q}}_{\mathrm{bw}}$, $\widetilde{h}^{\mathrm{Q}}_{\mathrm{fw}}$ and $\widetilde{h}^{\mathrm{Q}}_{\mathrm{bw}}$ can be combined into bi-directional symbolic search, but as in the previous cases, it rarely pays off to use the path-dependent variant of the heuristics. Table A.3 summarizes the comparison on selected variants. Overall, we observe that using the consistent state-dependent operator-potential heuristics is mostly beneficial compared to the same heuristics without normalizing the task.

References

[1] V. Alcázar, The consistent case in bidirectional search and a bucket-to-bucket algorithm as a middle ground between front-to-end and front-to-front, in: Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21), 2021, pp. 7–15.

[2] V. Alcázar, D. Borrajo, S. Fernández, R. Fuentetaja, Revisiting regression in planning, in: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13), 2013, pp. 2254–2260.

[3] V. Alcázar, P.J. Riddle, M. Barley, A unifying view on individual bounds and heuristic inaccuracies in bidirectional search, in: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20), 2020, pp. 2327–2334.

[4] V. Alcázar, Á. Torralba, 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), 2015, pp. 2–6.
[5] C. Bäckström, B. Nebel, Complexity results for SAS^+ planning, Comput. Intell. 11 (1995) 625–655.
[6] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans. Comput. 35 (1986) 677–691.
[7] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic model checking: 10^20 states and beyond, Inf. Comput. 98 (1992) 142–170.
[8] G. Ciardo, R. Siminiceanu, Using edge-valued decision diagrams for symbolic generation of shortest paths, in: Formal Methods in Computer-Aided Design, 4th International Conference (FMCAD'02), 2002, pp. 256–273.
[9] S. Edelkamp, Planning with pattern databases, in: Proceedings of the 6th European Conference on Planning (ECP'01), 2001, pp. 13–24.
[10] S. Edelkamp, Symbolic pattern databases in heuristic search planning, in: Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02), 2002, pp. 274–283.
[11] S. Edelkamp, Automated creation of pattern database search heuristics, in: Proceedings of the 4th Workshop on Model Checking and Artificial Intelligence (MoChArt 2006), 2006, pp. 35–50.
[12] S. Edelkamp, M. Helmert, Exhibiting knowledge in planning problems to minimize state encoding length, in: Proceedings of the 5th European Conference on Planning (ECP'99), 1999, pp. 135–147.
[13] S. Edelkamp, P. Kissmann, 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), 2008, pp. 1452–1453.
[14] S. Edelkamp, P. Kissmann, Optimal symbolic planning with action costs and preferences, in: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), 2009, pp. 1690–1695.
[15] S. Edelkamp, P. Kissmann, On the complexity of BDDs for state space search: a case study in connect four, in: Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11), 2011, pp. 18–23.
[16] S. Edelkamp, P. Kissmann, Á. Torralba, Symbolic A* search with pattern databases and the merge-and-shrink abstraction, in: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12), 2012, pp. 306–311.
[17] S. Edelkamp, P. Kissmann, Á. Torralba, BDDs strike back (in AI planning), in: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15), 2015, pp. 4320–4321.
[18] S. Edelkamp, F. Reffel, OBDDs in heuristic search, in: Proceedings of the 22nd Annual German Conference on Advances in Artificial Intelligence (KI'98), in: Lecture Notes in Computer Science, vol. 1504, Springer, 1998, pp. 81–92.
[19] D. Fišer, Lifted fact-alternating mutex groups and pruned grounding of classical planning problems, in: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20), 2020, pp. 9835–9842.
[20] D. Fišer, R. Horčík, A. Komenda, Strengthening potential heuristics with mutexes and disambiguations, in: Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS'20), 2020, pp. 124–133.
[21] D. Fišer, Álvaro Torralba, J. Hoffmann, Operator-potential heuristics for symbolic search, in: Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI'22), 2022, pp. 9750–9757.
[22] D. Fišer, Álvaro Torralba, J. Hoffmann, Operator-potentials in symbolic search: from forward to bi-directional search, in: Proceedings of the 32nd International Conference on Automated Planning and Scheduling (ICAPS'22), 2022, pp. 80–89.
[23] D. Fišer, A. Komenda, Fact-alternating mutex groups for classical planning, J. Artif. Intell. Res. 61 (2018) 475–521.
[24] S. Franco, L.H.S. Lelis, M. Barley, The Complementary2 planner in IPC 2018, in: IPC 2018 Planner Abstracts, 2018, pp. 32–36.
[25] S. Franco, A. Torralba, L.H. Lelis, M. Barley, On creating complementary pattern databases, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17), 2017, pp. 4302–4309.
[26] E.A. Hansen, R. Zhou, Z. Feng, Symbolic heuristic search using decision diagrams, in: S. Koenig, R.C. Holte (Eds.), Abstraction, Reformulation and Approximation, 5th International Symposium, Proceedings, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, in: Lecture Notes in Computer Science., vol. 2371, Springer, 2002, pp. 83–98.
[27] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107.
[28] P. Haslum, H. Geffner, Admissible heuristics for optimal planning, in: Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00), 2000, pp. 140–149.
[29] M. Helmert, C. Domshlak, Landmarks, critical paths and abstractions: what's the difference anyway?, in: Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS'09), 2009, pp. 162–169.
[30] M. Helmert, P. Haslum, J. Hoffmann, R. Nissim, Merge & shrink abstraction: a method for generating lower bounds in factored state spaces, J. Assoc. Comput. Mach. 61 (2014) 16:1–16:63.
[31] R.C. Holte, A. Felner, G. Sharon, N.R. Sturtevant, J. Chen, MM: a bidirectional search algorithm that is guaranteed to meet in the middle, Artif. Intell. 252 (2017) 232–266.
[32] R.M. Jensen, R.E. Bryant, M.M. Veloso, SetA*: an efficient BDD-based heuristic search algorithm, in: Proceedings of the 18th National Conference of the American Association for Artificial Intelligence (AAAI'02), 2002, pp. 668–673.
[33] R.M. Jensen, M.M. Veloso, R.E. Bryant, State-set branching: leveraging BDDs for heuristic search, Artif. Intell. 172 (2008) 103–139.
[34] P. Kissmann, S. Edelkamp, Improving cost-optimal domain-independent symbolic planning, in: Proceedings of the 25th National Conference of the American Association for Artificial Intelligence (AAAI'11), 2011, pp. 992–997.
[35] Y. Lai, M. Pedram, S.B.K. Vrudhula, Formal verification using edge-valued binary decision diagrams, IEEE Trans. Comput. 45 (1996) 247–255.
[36] A. Martelli, On the complexity of admissible search algorithms, Artif. Intell. 8 (1977) 1–14.
[37] K.L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.
[38] F. Pommerening, M. Helmert, A normal form for classical planning tasks, in: Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15), 2015, pp. 188–192.
[39] F. Pommerening, M. Helmert, B. Bonet, Higher-dimensional potential heuristics for optimal classical planning, in: S. Singh, S. Markovitch (Eds.), Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17), 2017, pp. 3636–3643.
[40] F. Pommerening, M. Helmert, G. Röger, J. Seipp, From non-negative to general operator cost partitioning, in: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15), 2015, pp. 3335–3341.
[41] F. Pommerening, G. Röger, M. Helmert, B. Bonet, LP-based heuristics for cost-optimal planning, in: Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14), 2014, pp. 226–234.
[42] J. Seipp, Fast downward scorpion, in: IPC 2018 Planner Abstracts, 2018, pp. 77–79.
[43] J. Seipp, M. Helmert, Counterexample-guided Cartesian abstraction refinement for classical planning, J. Artif. Intell. Res. 62 (2018) 535–577.
[44] J. Seipp, F. Pommerening, M. Helmert, New optimization functions for potential heuristics, in: Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15), 2015, pp. 193–201.
[45] J. Seipp, F. Pommerening, G. Röger, M. Helmert, Correlation complexity of classical planning domains, in: Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16), 2016, pp. 3242–3250.
[46] E. Shaham, A. Felner, N.R. Sturtevant, J.S. Rosenschein, Optimally efficient bidirectional search, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI'19), 2019, pp. 6221–6225.

[47] S.S. Shperberg, A. Felner, N.R. Sturtevant, S.E. Shimony, A. Hayoun, Bidirectional heuristic search: expanding nodes by a lower bound, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20), 2020, pp. 4775–4779.
[48] S. Sievers, M. Helmert, Merge-and-shrink: a compositional theory of transformations of factored transition systems, J. Artif. Intell. Res. 71 (2021) 781–883.
[49] S. Sievers, M. Wehrle, M. Helmert, An analysis of merge strategies for merge-and-shrink heuristics, in: Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16), 2016, pp. 294–298.
[50] D. Speck, F. Geißer, R. Mattmüller, Symbolic planning with edge-valued multi-valued decision diagrams, in: Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS'18), 2018, pp. 250–258.
[51] D. Speck, F. Geißer, R. Mattmüller, 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), 2020, pp. 263–271.
[52] D. Speck, M. Katz, Symbolic search for oversubscription planning, in: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21), 2021, pp. 11972–11980.
[53] D. Speck, R. Mattmüller, B. Nebel, Symbolic top-k planning, in: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20), 2020, pp. 9967–9974.
[54] Á. Torralba, V. Alcázar, Constrained symbolic search: on mutexes, BDD minimization and more, in: Proceedings of the 6th Annual Symposium on Combinatorial Search (SOCS'13), 2013, pp. 175–183.
[55] Á. Torralba, V. Alcázar, D. Borrajo, P. Kissmann, S. Edelkamp, SymBA*: a symbolic bidirectional A* planner, in: IPC 2014 Planner Abstracts, 2014, pp. 105–109.
[56] Á. Torralba, V. Alcázar, P. Kissmann, S. Edelkamp, cGamer: constrained gamer, in: IPC 2014 Planner Abstracts, 2014.
[57] Á. Torralba, V. Alcázar, P. Kissmann, S. Edelkamp, Efficient symbolic search for cost-optimal planning, Artif. Intell. 242 (2017) 52–79.
[58] Á. Torralba, S. Edelkamp, P. Kissmann, Transition trees for cost-optimal symbolic planning, in: Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13), 2013, pp. 206–214.
[59] Á. Torralba, C. Linares López, D. Borrajo, Abstraction heuristics for symbolic bidirectional search, in: Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16), 2016, pp. 3272–3278.
[60] Á. Torralba, C. Linares López, D. Borrajo, Symbolic perimeter abstraction heuristics for cost-optimal planning, Artif. Intell. 259 (2018) 1–31.