On Bidirectional Heuristic Search in Classical Planning: An Analysis of BAE*

Kilian Hu,[1] **David Speck**[1,2]

[1]University of Freiburg, Freiburg, Germany
[2]Linköping University, Linköping, Sweden
kilian@hu101.com, speckd@informatik.uni-freiburg.de

Abstract

Heuristic search is a successful approach to cost-optimal planning. Bidirectional heuristic search algorithms have been around for a long time, but only recent advances have led to algorithms like BAE* that have the potential to outperform unidirectional heuristic search algorithms like A* in practice. In this work, we analyze BAE* for classical planning and the challenges associated with the underlying assumption of an explicit state representation. We show that it is crucial to use mutexes and reachability analysis to reduce the potentially exponential number of goal states, which makes it possible to create an explicit representation of a reversed planning task that can be used for the backward search of BAE*. Our empirical evaluation shows that BAE* solves more instances than A* in multiple domains with significantly fewer node expansions, demonstrating the usefulness of BAE* in planning.

Introduction

Heuristic search has been one of the dominant approaches to classical planning in recent decades. In cost-optimal planning, (forward) A* (Hart, Nilsson, and Raphael 1968) is the most prominent heuristic search algorithm. While A* explores the state space unidirectionally, searching either from the initial state to a goal state (forward A*) or from a goal state to the initial state (backward A*), front-to-end bidirectional heuristic search performs these two searches simultaneously, stopping when both meet while satisfying a certain termination condition (Pohl 1969). Although bidirectional heuristic search algorithms can in theory outperform unidirectional heuristic search algorithms, they have almost never performed much better empirically (Barker and Korf 2015). This observation, together with the drawbacks of backward search in classical planning caused by the presence of partial states where some values of the state variables are unknown (Alcázar et al. 2013), are the reasons why bidirectional heuristic search for classical planning has received little attention.

A notable exception in classical planning is symbolic search, where bidirectional search is considered the dominant search strategy (Torralba et al. 2017; Speck, Geißer, and Mattmüller 2020). While there have been recent successes in combining symbolic search with heuristic search (Fišer, Torralba, and Hoffmann 2022), most modern symbolic planners are mainly based on bidirectional symbolic *blind* search, which achieves state-of-the-art performance (Edelkamp, Kissmann, and Torralba 2015) without using any heuristic functions (Kissmann, Edelkamp, and Hoffmann 2014; Torralba et al. 2014; Speck, Geißer, and Mattmüller 2018).

More recently, the interest in bidirectional heuristic search has been rekindled due to improved theoretical understanding which lead the development of new successful algorithms with strong theoretical properties (Barker and Korf 2015; Eckerle et al. 2017; Holte et al. 2017; Chen et al. 2017; Alcázar, Riddle, and Barley 2020; Alcázar 2021). Alcázar, Riddle, and Barley (2020) highlighted the strong performance of a best-first bidirectional search algorithm based on heuristic errors called BAE* (Sadhukhan 2012, 2013), by performing a comparison with other bidirectional search algorithms. Interestingly, BAE* had a strong performance, even compared to more sophisticated algorithms that incorporate the ideas of BAE*. Therefore, BAE* is a promising candidate to investigate whether and how recent advances in bidirectional heuristic search with strong empirical results can be transferred to classical planning.

In this paper, we analyze BAE* for classical planning in theory and practice. The content of this paper is organized as follows. First, we introduce the relevant background for this paper concerning classical planning and heuristic search. Next, we explain BAE* and highlight the challenges associated with the underlying assumption of an explicit state representation when using BAE* for classical planning. To make the backward search of BAE* possible, we define a reversed planning task that does not rely on partial states, but has a potentially exponential increase in size. We show that the use of mutexes and reachability analysis can significantly reduce the number of goal states, making it possible to create the reversed task in practice for many domains. An empirical evaluation comparing A* in both forward and backward directions with BAE* shows that A* performs best overall, but BAE* solves more instances than A* in several domains with significantly fewer node expansions. A simple portfolio approach that selects BAE* if the planning task is reversible within reasonable resource bounds and selects A* otherwise is already sufficient to outperform A*, proving the usefulness of bidirectional heuristic search in classical planning. Finally, we discuss future research directions and draw a conclusion.

Background

We consider classical planning tasks as defined by the SAS$^+$ formalism (Bäckström and Nebel 1995). A *planning task* is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_I, s_\star \rangle$ consisting of four components. $\mathcal{V}$ is a finite set of *state variables* where each variable $v \in \mathcal{V}$ has a finite *domain* $D_v = \{0, \dots, |D_v| - 1\}$. A *fact* is a pair (v, d), where $v \in \mathcal{V}$ and $d \in D_v$ and a *partial state* s is a consistent set of facts, i.e., each variable occurs in at most one of the facts in s. If s assigns a value to each variable $v \in \mathcal{V}$, s is called a *state*. We refer to the set of all possible states over $\mathcal{V}$ as S. (Partial) states can be seen as (partial) functions which map variables to values, i.e., $s[v]$ is the value of variable v in state s. We denote the set of all variables contained in a (partial) state as $vars(s)$. $\mathcal{O}$ is a finite set of *operators*, where for an operator $o \in \mathcal{O}$ the *precondition* $pre(\text{o})$ and *effect* $\mathit{eff}(\text{o})$ both are partial states. We refer to the variables which occur in $pre(\text{o})$ as $prevars(\text{o})$ as a shorthand for $vars(pre(\text{o}))$ (analogously $\mathit{effvars}(\text{o})$ for the effect). An operator $o \in \mathcal{O}$ is *applicable* in state s iff $pre(o) \subseteq s$. If applicable, applying o in s yields the state s' where $s'[v] = \mathit{eff}(o)[v]$ for all $v \in \mathit{effvars}(o)$ and $s'[v] = s[v]$ otherwise. We write $s[\![o]\!]$ for s'. Each operator o has non-negative *cost* written as $cost(o) \in \mathbb{N}_0$. The state $s_I \in S$ is called the *initial state* and the partial variable assignment $s_\star$ specifies the *goal condition* that defines all possible *goal states* $S_\star \subseteq S$.

The objective of classical planning is to determine a *plan* $\pi = \langle o_0, \dots, o_{n-1} \rangle$ that is a sequence of operators $o_i \in \mathcal{O}$. We call π applicable in $s_0 \in S$ if there exist states $s_1, \dots, s_n$ such that o_i is applicable in s_i and $s_{i+1} = s_i[\![o_i]\!]$ for all $i = 0, \dots, n-1$. We call π a plan for Π if it is applicable in s_I and if $s_n \in S_\star$. The *cost* of a plan π is the sum of operator costs, i.e., $cost(\pi) = \sum_{i=0}^{n-1} cost(o_i)$. While the objective of satisficing planning is to find any plan for Π, we focus on optimal planning, where the objective is to find an *optimal plan*, i.e., a plan π for which there is no plan π' with $cost(\pi') < cost(\pi)$.

We consider the *delete relaxation* of a task Π, which we denote as Π^+, where the operators do not have a "negative" effect (Bonet and Geffner 2001). More precisely, the application of a *relaxed operator* o^+ only adds the effect to the relaxed state, i.e., $s^+[\![o^+]\!] = s^+ \cup \mathit{eff}(o^+)$. Consequently, the requirement that the states have to be consistent is omitted, i.e., an *relaxed state* s^+ can contain multiple facts with the same variable. A pair of facts is called *mutually exclusive (mutex)* if there is no reachable state $s \in S$ from s_I such that s contains both facts. Finally, a state is called *spurious* if it cannot be reached from the initial state.

Heuristic Search

A* (Hart, Nilsson, and Raphael 1968) is a widely used heuristic search algorithm for solving planning problems. A* builds upon Dijkstra's algorithm (Dijkstra 1959) by making use of a heuristic function that guides the search towards promising parts of the state space. A *heuristic* is a computable function $h : S \to \mathbb{N}_0 \cup \{\infty\}$ that, given a state $s \in S$, estimates the cost of reaching a state in $S_\star$ from s (Pearl 1984). The perfect heuristic h^* maps each state s to the cost of the cheapest path from s to any goal state. If the goal cannot be reached from state s, then $h^*(s) = \infty$. A heuristic is admissible iff it never overestimates the cost of reaching a goal state, i.e., $h(s) \leq h^*(s)$ for all $s \in S$. A heuristic is consistent iff it is admissible and $h(s) \leq h(s[\![o]\!]) + cost(o)$ for all states $s \in S$ and operators $o \in \mathcal{O}$ applicable in s. Since BAE* assumes consistent heuristics (Alcázar, Riddle, and Barley 2020), from now on we consider only consistent heuristics.

To store information about states, A* uses (search) nodes. A *node* n refers to a single state s, has a reference to its *parent* node from which is was created and stores several values like the cost of the path leading to the node, denoted as $g(n)$ and the heuristic value of the node's state, denoted as $h(n)$. A* maintains an *open list* that initially contains only one node with the initial state, and a *closed list* to keep track of the states already seen. Iteratively, A* selects from the open list the node n that minimizes the priority function $f(n) = g(n) + h(n)$, and expands n. The *expansion* of n, adds the state s associated with n to the closed list and generates nodes for all possible *successors* $succ = \{s[\![o]\!] \mid o \in \mathcal{O}, pre(o) \subseteq s\}$ of s, from which those states not yet seen, i.e., not part of the closed list, are added to the open list. A solution is found when a goal node is expanded and given a consistent heuristic the found solution is optimal.

Since we are interested in bidirectional heuristic search, where separate forward and backward search is performed, we label the values with the corresponding search direction, e.g., a forward heuristic estimate is written as $h_f(n)$ and the g-value of a node in the backward direction is written as $g_b(n)$. If the direction of a value can be either forward or backward, the value is denoted by x, e.g., f_x. The opposite direction is called $\bar{x}$. Note that the exact realization of backward search for a given classical planning task is explained in detail in the next section.

BAE* in Planning

BAE* (Sadhukhan 2012) is a bidirectional best-first search algorithm which is based on heuristic errors. In the following, we explain the functioning of BAE* as described in Alcázar, Riddle, and Barley (2020). We then discuss how backward search of BAE* is possible for a classical planning task, and discuss the challenges involved as well as possible solutions and improvements for better performance.

Similar to other bidirectional heuristic search algorithms, bidirectional heuristic search using error estimate *(BAE*)* performs two A*-like searches, one in the forward direction and one in the backward direction. The core idea of BAE* is to use a priority function for the open lists $Open_x$ that incorporates information from the other search direction $\bar{x}$ by computing the heuristic errors accumulated along a path. The *forward error* along a path is defined as $FE_x(n) = g_x(n) + h_x(n) - h_x(n_{I,x})$, where $n_{I,x}$ is the node associated

Algorithm 1: Pseudocode of BAE* (inspired by Sadhukhan (2012) and Alcázar, Riddle, and Barley (2020)).

```
 1: $Open_f \leftarrow \{\text{makeInitNode}_f(s_{I,f})\}$;
 2: $Open_b \leftarrow \{\text{makeInitNode}_b(s_{I,b})\}$;
 3: $Closed_f \leftarrow \emptyset$; $Closed_b \leftarrow \emptyset$;
 4: $L \leftarrow 0$; $U \leftarrow \infty$; $\pi \leftarrow$ no plan; $x \leftarrow f$;
 5: while $Open_f \neq \emptyset$ and $Open_b \neq \emptyset$ do
 6:    $bMin_f \leftarrow Open_f.\text{minPriority}()$
 7:    $bMin_b \leftarrow Open_b.\text{minPriority}()$
 8:    $L \leftarrow \frac{bMin_f + bMin_b}{2}$
 9:    $n \leftarrow Open_x.\text{popMin}()$
10:    $Closed_x.\text{insert}(n)$
11:    if $n \in Open_{\bar{x}}$ and $g_x(n) + g_{\bar{x}}(n) < U$ then
12:       $U \leftarrow g_x(n) + g_{\bar{x}}(n)$
13:       $\pi \leftarrow \text{extract-plan}(n)$
14:    end if
15:    if $L \geq U$ then
16:       return $\pi$
17:    end if
18:    for $n' \in succ_x(n) \setminus Closed_x$ do
19:       $b_x(n') \leftarrow f_x(n') + d_x(n')$
20:       $Open_x.\text{insert}(node\colon n', priority\colon b_x(n'))$
21:    end for
22:    $x \leftarrow \text{chooseDirection}()$
23: end while
24: return $\pi$
```

with the initial state of direction x.[1] The *backward error* is defined as $BE_x(n) = g_x(n) - h_{\bar{x}}(n)$ and the *total error* is defined as $TE_x(n) = FE_x(n) + BE_x(n)$. Alcázar, Riddle, and Barley (2020) note that the heuristic value of the initial state in the forward error in the priority function can be omitted since it is a constant. This yields a bidirectional estimate of a node n, the *b-value*, defined as $b_x(n) = f_x(n) + d_x(n)$ where $d_x(n) = g_x(n) - h_{\bar{x}}(n)$ is the heuristic inaccuracy (Kaindl and Kainz 1997; Alcázar, Riddle, and Barley 2020). Intuitively, the *d-value* $d_x(n)$ of a node n describes the actual error that the backward heuristic makes in evaluating $h_{\bar{x}}(n)$ (Alcázar, Riddle, and Barley 2020). In BAE*, the priority function uses the b-value $b_x(n)$ to assign a node n a priority value in search direction x.

Finally, the termination criterion of BAE* considers a lower and upper bound on the cost of the optimal solution, denoted by L and U, respectively. The *lower bound* L is defined as $L = \frac{bMin_f + bMin_b}{2}$ where $bMin_x$ is the minimum b-value of a node in the open list $Open_x$. The *upper bound* U is the cost of the best solution found so far. BAE* terminates and returns the best plan found if the lower bound exceeds the upper bound, i.e., $L \geq U$, or if one of the open lists is empty.

Algorithm 1 shows the pseudocode of BAE*. First, the open and closed lists are initialized for both search directions (lines 1 to 3). In line 4, the lower bound L, the upper bound U and the best plan found π are initialized and the search direction x is initially set to forward. The algorithm continues as long as no open list is empty (line 5) or the lower bound exceeds the upper bound (line 15), in both cases the best plan found is returned (lines 16 and 24). In lines 6 to 8, the lower bound L is updated and in lines 9 and 10, the most promising node n in the chosen direction x is extracted from the corresponding open list and added to the closed list. Then we check if the search frontiers have met and if so, if we have found a better solution, which then leads to an update of the upper bound and the best plan found so far (lines 11 to 14). In lines 18 to 21, the successor states of n are generated taking into account the search direction x, and all nodes not included in the closed list are added to the corresponding open list. Finally, the next search direction is selected in line 22. The original BAE* algorithm uses an alternating direction selection (Sadhukhan 2012), but in general all strategies that iteratively switch between both search directions guarantee optimality if the termination criterion is chosen appropriately.

[1]Note that BAE* assumes a single goal state, which is considered as the initial state in the backward direction.

Backward Search

BAE* is a bidirectional search algorithm that performs a forward and backward search. To perform a backward search on a given planning task, it is natural to define a *reversed* (or *backward*) version of the given task. There are several methods in the literature for generating such a reversed planning task that enables backward search, such as the creation of the dual of a PDDL planning task (Suda 2014), the creation of a regression state model of a STRIPS planning task (Geffner and Bonet 2013), or, similar to our definition, the creation of a reversed SAS$^+$ planning task (Alcázar et al. 2013). A common definition of a reversed SAS$^+$ planning task and the subsequent search use partial states (Alcázar et al. 2013) to avoid a potential exponential increase in the size of the reversed task that can arise from the partial goal state, since it can describe exponentially many goal states. However, partial states affect the performance of a bidirectional search due to the non-trivial intersection check of the two search frontiers. Recent bidirectional approaches (Kuroiwa and Fukunaga 2020) for satisficing planning even make tradeoffs between intersection testing completeness and better performance. Finally, and most importantly, applying BAE* to planning without changing the core algorithm requires an explicit state representation.

Therefore, we propose to construct a reversed SAS$^+$ planning task suitable for BAE* with an explicit state representation, potentially increasing the task size exponentially. In the empirical evaluation, we find that it is possible to quickly determine whether we can construct such a reversed planning task for a given planning task, and if so, BAE* often provides better performance than forward A*. Modifying BAE* to allow for searching with a more compact representation using partial states is an interesting line of research that we leave open for future work.

Definition 1 (Reversed Task)**.** *Given a planning task* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_I, s_\star \rangle$*, we define a* reversed planning task *as* $reversed(\Pi) = \Pi_b = \langle \mathcal{V}, \mathcal{O}_b, s_\star, s_I \rangle$.

There are two critical aspects to discuss regarding the construction and handling of the reversed planning task Π_b, namely 1) that Π_b has a partial initial state $s_\star$ and 2) how the set of operators $\mathcal{O}_b$ is defined.

Multiple Initial States In general, it is possible to use a dummy variable and several dummy goal operators to introduce a unique dummy goal state for a given planning task Π such that Π_b has a unique initial state. However, a closer look at the reversed task Π_b generated by this procedure reveals that the operators leading from such a dummy goal state to the actual goal states $S_\star$ are only relevant in the first expansion of a search algorithm. Moreover, the presence of such dummy operators and a dummy variable can negatively affect the informativeness of heuristics. Therefore, instead of generating these operators, we propose to generate all goal states of $s_\star$ and insert them directly into the backward open list during initialization to save the memory and time required to generate these operators. Clearly, there can be exponentially many goal states specified by $s_\star$, leading to exponentially many initial states to be generated. Therefore, we propose to use mutexes and a reachability analysis to reduce the number of possible initial states of Π_b.

Given a set of mutexes, the computation of all valid goal states of Π, i.e., all valid initial states of Π_b, can be described as a search for all solutions to a Constraint Satisfaction Problem (CSP) (Russell and Norvig 2003), where the variables of the CSP are all variables not specified in the goal condition with their respective domains and the mutexes represent the constraints. To solve this CSP, we assign values to the variables in a fixed order and check after each assignment whether any mutex has been violated. In case of violation, we stop trying to complete the current assignment and try the next value for the current variable. This backtracking procedure avoids a lot of unnecessary computations.

In addition, we propose to reduce the set of goal states by performing a reachability analysis in the delete relaxed reversed task, i.e., we discard a goal state if the initial state cannot be reached from it by going backwards in the delete relaxation. While this reduction of the number of goal states can lead to noticeable overhead, it is particularly beneficial for sampling-based heuristics such as the diverse-potential heuristic (Seipp, Pommerening, and Helmert 2015), since the samples are drawn from more relevant parts of the search space, which increases the quality of the heuristic.

Reversed Operators We define the set of reversed operators as $\mathcal{O}_b = \bigcup_{o \in \mathcal{O}} \mathit{reverse}(o)$, where $\mathit{reverse}(o)$ yields the set of reversed operators for each operator $o \in \mathcal{O}$. To generate the set of reversed operators $\mathit{reverse}(o)$ for an operator $o \in \mathcal{O}$, we consider three cases for each variable $v \in \mathit{prevars}(o) \cup \mathit{effvars}(o)$:

1) If $v \in \mathit{prevars}(o) \land v \in \mathit{effvars}(o)$, we can simply swap the precondition and the effect of these variables, i.e., $\mathit{pre}(o_b)[v] = \mathit{eff}(o)[v]$ and $\mathit{eff}(o_b)[v] = \mathit{pre}(o)[v]$ for all $o_b \in \mathit{reverse}(o)$.
2) If $v \in \mathit{prevars}(o) \land v \notin \mathit{effvars}(o)$, the reversed operators keep that same precondition, i.e., $\mathit{pre}(o_b)[v] = \mathit{pre}(o)[v]$ for all $o_b \in \mathit{reverse}(o)$.
3) If $v \notin \mathit{prevars}(o) \land v \in \mathit{effvars}(o)$, more work is needed because after applying the operator o it is unclear what value the variable v had before. Thus, such an operator does not lead to a single reversed operator, but to a set of reversed operators. We collect all variables belonging to this case and for each possible assignment of these variables we create a corresponding operator that has the particular assignment among its effect. We note that the number of possible assignments for these variables grows exponentially with the number of variables.

This method of generating a reversed task makes it possible to readily apply BAE* to a classical planning task. However, this method has the disadvantage that the size of the reversed planning task is potentially exponential and that a large number of spurious states can affect the performance of BAE*.

Further Optimizations

In the following, we discuss further optimizations and design decisions that can improve the performance of BAE* for classical planning and that we use in our implementation for the empirical evaluation.

Mutex Pruning We prune states generated during the search which violate a mutex (Alcázar and Torralba 2015). Similar to backward heuristic search (Alcázar et al. 2013), this is especially important in the backward direction of BAE* to reduce the number of spurious states. Although it is in general not possible to generate all mutexes and prune all spurious states, in practice it is often possible to obtain a reasonable number of mutexes, which greatly improves performance (Alcázar and Torralba 2015).

Trimming and Screening We augment BAE* with two techniques, called *trimming* and *screening*, introduced with the bidirectional heuristic search algorithm BS* (Kwa 1989). Both techniques discard a node n if it cannot be part of an optimal solution because its estimated cost is higher than the cost of the best plan found so far, i.e., if $f(n) > U$. *Trimming* is applied each time U decreases, and accordingly discards nodes from the open lists, resulting in potentially lower memory consumption. *Screening* is applied before a node is inserted into an open list which can avoid the unnecessary cost of insertion. For BAE*, we adapt the delaying rule of the individual b bound from Definition 7 in Alcázar, Riddle, and Barley (2020), to define the *trimming* and *screening* criterion as $b_x(n) \geq 2U - \mathit{bMin}_{\bar{x}}$, which is different from the criterion used by Kwa (1989). Any node satisfying this inequality cannot be part of a better solution than the current one and can therefore be ignored.

Search Direction In the original implementation of BAE* (Sadhukhan 2012, 2013; Alcázar, Riddle, and Barley 2020), the two search directions are selected alternatingly. As with other bidirectional search algorithms, our experiments show that Pohl's cardinality criterion (Pohl 1971), i.e., expansion in the direction with the smaller open list, performs better overall.

Tie Breaking In forward A* and its backward version A$_b^*$, the strategy for breaking ties between the most promising states (smallest f-value) is usually to favor the states with smaller g-values. Our experiments for BAE* show that breaking ties in favor of larger g-values usually leads to better results. This observation is consistent with empirical results for other bidirectional heuristic search algorithms (Chen et al. 2017).

Empirical Evaluation

We implemented BAE* in Fast Downward Release 20.06 (Helmert 2006) and used the h2-based preprocessor of Alcázar and Torralba (2015) to generate mutexes for pruning spurious states and operators. For symbolic blind search, we used the symbolic planner SymK (Speck, Mattmüller, and Nebel 2020; Speck 2022). All of our benchmarks, source code and experiment data are available online (Hu and Speck 2022). For the empirical evaluation we used a maximum of 4 GB memory and 30 minutes runtime. All experiments were run on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB of cache, and 188 GB of shared memory, running Ubuntu 20.04.3 LTS 64 bit using Downward Lab (Seipp et al. 2017). Our benchmark suite contains the 1816 planning instances from the optimal tracks of the International Planning Competitions (IPCs), excluding unsolvable instances or instances with axioms or conditional effects.

In what follows, we analyze various aspects of BAE* and compare it with forward A* and backward A$_b^*$. We use three prominent representatives of consistent heuristics in classical planning, namely the relaxation heuristic $h^{\max}$ (Bonet and Geffner 2001), the incremental pattern database heuristic h^{iPDB} (Haslum et al. 2007) and the diverse potentials heuristic h^{pot} (Seipp, Pommerening, and Helmert 2015). We limit the hill climbing time for h^{iPDB} to 5 minutes for each of the involved search directions and use CPLEX release 12.10 as a linear program solver for h^{pot}. Next and first, we analyze whether and how often it is possible to generate a reversed task (Definition 1) that can be readily used for the backward search of BAE* and A$_b^*$.

Reversed Task Generation

We analyze the feasibility of generating a reversed task with an explicit state representation to perform BAE* readily. The generation of the reversed planning task in a naive way, i.e., without pruning the number of goal states, is possible only for 833 out of 1816 planning instances within the given time and memory limit. The main reason for this is the high number of goal states that can occur due to the partial goal state definition of classical planning tasks. Interestingly, reversing operators generally turns out to be unproblematic, with an average of 1.44 reversed operators generated per operator across all instances. When pruning the set of goal states using the methods discussed, i.e., using mutexes and a reachability analysis of the delete relaxed reversed task, it is possible to create the reversed planning task of 1256 planning instances. Moreover, as shown in Figure 1, the number of goal states is greatly reduced, which has a direct positive impact on the performance of BAE* and some (sampling-based) heuristics.

Overall, the time to generate reversed planning tasks heavily depends on the domain and the instance size. The average time to generate a reversed planning task using the pruning techniques discussed is about 16 seconds, which is feasible, but sometimes reversing the task along with the pruning techniques requires too much time or memory, resulting in 560 instances for which BAE* cannot be carried out. Clearly, this issue needs to be tackled in the future, which we will go into in more detail in the Discussion.

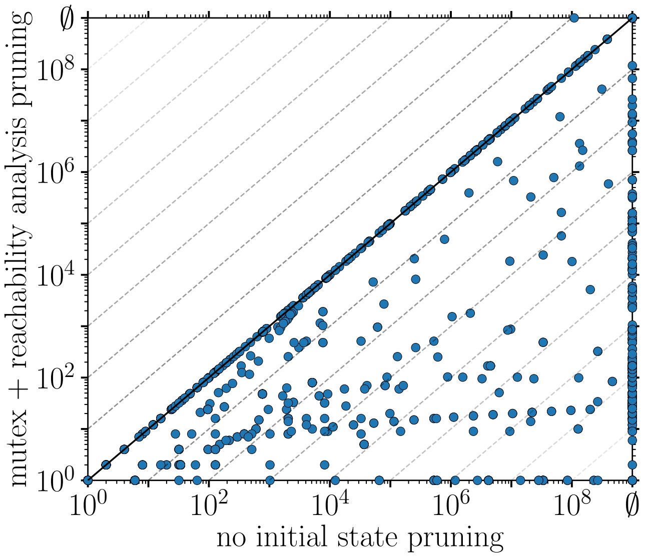

Figure 1: Comparison of the number of initial states of the reversed planning task with and without mutex and reachability analysis pruning. With $\emptyset$ we denote instances for which the reversed planning tasks could not be generated within the time and memory limits.

Search Performance

The natural question is how A* compares to BAE* in terms of search performance on classical planning tasks. To evaluate the search performance, we compare the coverage, i.e., the sum of optimally solved tasks, the number of node expansions, and the runtime of the considered heuristic search algorithms.

Coverage Table 1 shows a domain-wise comparison of the coverage of A*, A$_b^*$, and BAE* using three different heuristics. It can be observed that A* performs best overall for each heuristic. However, the first section of the Table 1 shows that there are domains where BAE* achieves higher coverage than A*, while the second section highlights domains where BAE* clearly falls behind A*. Interestingly, the performance difference between BAE* and A* (as well as A$_b^*$) is largest for the h^{iPDB} heuristic, due to the increased overhead of computing two abstraction heuristics in both forward and backward directions. Comparing the coverage of BAE* and A$_b^*$, we find that BAE* performs better overall for all three heuristics. Since both BAE* and A$_b^*$ require a reversed task, only the search performance is compared here, and the advantages of BAE* are clearly evident.

Comparing heuristic explicit search with blind symbolic search (Table 1), we find that bidirectional symbolic search

	$h^{\max}$			h^{iPDB}			h^{pot}			Symbolic Search		
	A*	A^*_b	BAE*	A*	A^*_b	BAE*	A*	A^*_b	BAE*	fw	bw	bd
barman (34)	**11**	4	6	**4**	0	**4**	4	4	**6**	11	12	**15**
blocksworld (35)	21	22	**30**	28	**32**	31	28	28	**30**	21	21	**30**
driverlog (20)	9	7	**11**	13	13	**14**	12	11	**13**	11	7	**12**
elevators (50)	**35**	12	29	**43**	32	38	28	14	**29**	31	14	**43**
ged (20)	15	15	**20**	19	13	**20**	19	13	**20**	15	10	**19**
logistics (63)	14	12	**17**	29	**31**	29	21	24	**25**	19	19	**23**
miconic (150)	**55**	**55**	**55**	**69**	60	66	55	55	**60**	101	**116**	**116**
parcprinter (50)	**37**	35	31	**45**	34	32	47	**48**	**48**	**39**	**39**	35
pegsol (50)	46	16	**48**	**50**	44	48	48	40	**50**	46	24	**48**
termes (20)	10	8	**15**	13	12	**16**	12	11	**16**	12	8	**18**
zenotravel (20)	**8**	7	**8**	**13**	12	12	10	8	**11**	8	8	**10**
data-network (20)	**11**	0	0	**12**	0	0	**9**	0	0	10	9	**13**
mprime (35)	**24**	1	1	**24**	1	1	**23**	1	1	**22**	8	**22**
mystery (19)	**17**	3	3	**17**	3	3	**17**	3	3	**15**	8	**15**
pipes-nt (50)	**20**	8	11	**21**	10	11	**24**	11	11	14	8	**15**
pipes-t (50)	**12**	4	5	**19**	5	5	**17**	5	5	15	6	**16**
snake (20)	**11**	0	0	**13**	0	0	**12**	0	0	**4**	0	**4**
tidybot (40)	**26**	1	1	**24**	1	1	**22**	1	1	14	9	**19**
other domains (1070)	**465**	348	415	**553**	429	474	**581**	455	512	553	534	**603**
Sum (1816)	**847**	558	706	**1009**	732	805	**989**	732	841	961	860	**1076**

Table 1: A domain-wise comparison between forward A*, backward A^*_b, and BAE* with three different heuristics and forward (fw), backward (bw) and bidirectional (bd) symbolic *blind* search. The maximum coverage for each heuristic and symbolic blind search are highlighted separately. We explicitly show coverage results for domains with an interesting coverage difference, where the first section are domains where BAE* performs favorably, and the second section are domains where BAE* clearly falls behind A*.

	$h^{\max}$			h^{iPDB}			h^{pot}		
	A*	A^*_b	BAE*	A*	A^*_b	BAE*	A*	A^*_b	BAE*
Sum (1256)	**719**	558	706	**853**	732	805	**848**	732	841

Table 2: Comparison of the overall coverage between forward A*, backward A^*_b, and BAE* with three different heuristics considering only tasks for which it was possible to generate the reversed task within the given time and memory limits.

is advantageous compared to forward and backward symbolic search almost in every domain, which is not the case with heuristic explicit search. The question arises whether this difference is due to the tasks for which it is not possible to generate the reversed task and thus BAE* cannot be applied. Looking at Table 2, which reports the overall coverage only considering the 1256 tasks for which it was possible to generate the reversed task within the given time and memory limits, shows that this is partly the reason, as the coverage gap between BAE* and A* gets smaller. Note that these coverage analyses are based on aggregate data and thus may not show the complementary strengths of A*, A^*_b and BAE* resulting in different levels of performance on different tasks, which we will analyze in more detail below.

Node Expansions To compare the actual search performance of BAE* with A* and A^*_b, we consider the number of node expansions performed until an optimal solution is found. Figures 2a, 2e and 2i compare the node expansions of BAE* with A*. Looking at the instances solved by both BAE* and A*, we see that BAE* requires significantly fewer node expansions than A* in several instances, while A* requires fewer expansions than BAE* in others. A different picture emerges when comparing the number of node expansions of BAE* and A^*_b, where the comparison shifts in favor of BAE* (Figures 2c, 2g, and 2k). Overall, the node expansion comparisons show mixed results, indicating that no algorithm strictly dominates another and all have their merits. However, it appears that when backward search is possible, BAE* often compares favorably to A^*_b.

Runtime Considering the overall runtime of the considered heuristic search algorithms, we can see that the trends of the node expansion comparison are also present here (Figures 2b, 2f, and 2j, and Figures 2d, 2h, and 2l.). This is not surprising, since the number of node expansions usually has

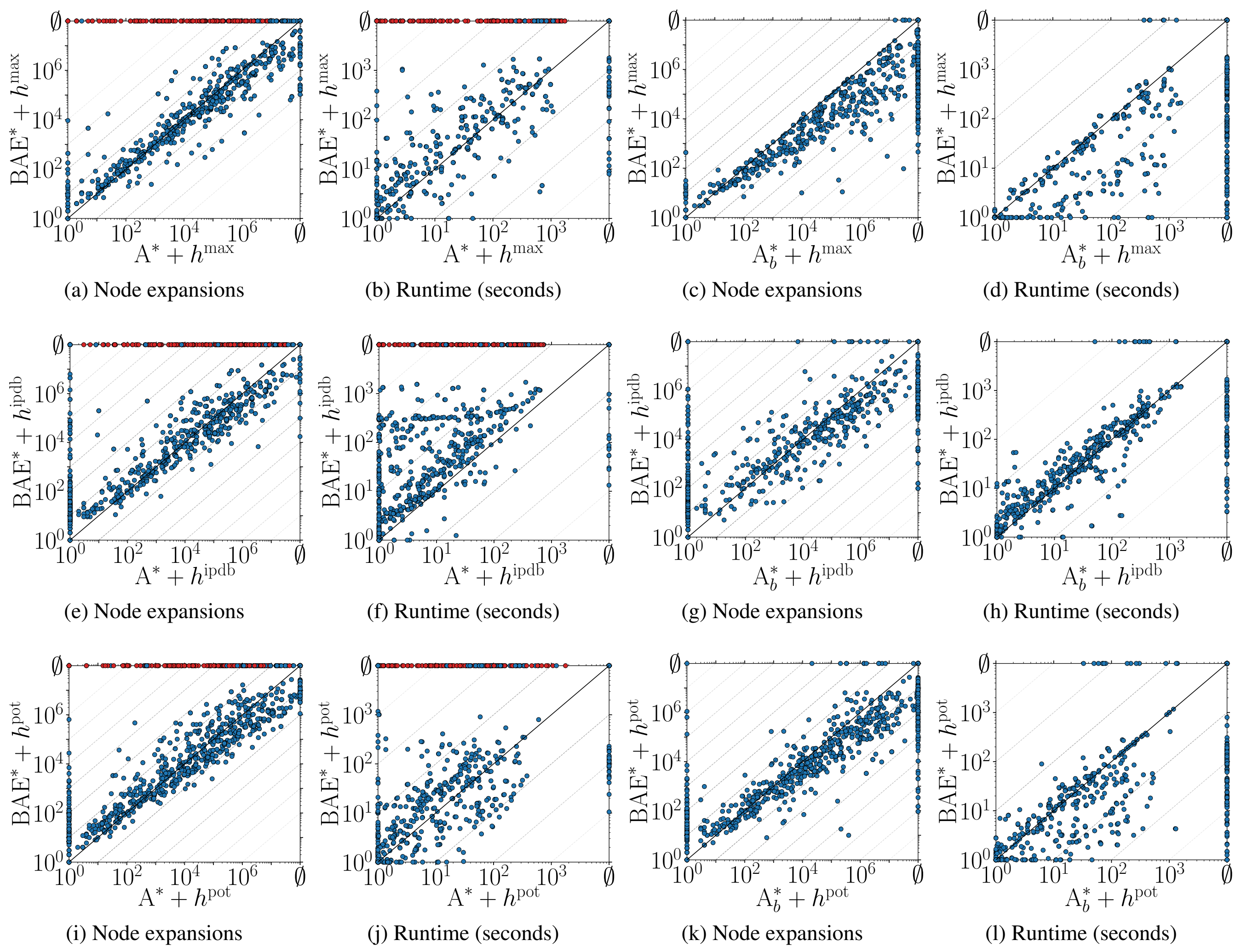

Figure 2: Comparison of BAE* with forward A* and backward A$_b^*$ using the max heuristic $h^{\max}$, the incremental pattern database heuristic h^{iPDB} and the diverse potentials heuristic h^{pot}. The runtime includes both the time needed to generate the reversed tasks (for A$_b^*$ and BAE*) and the actual search time. We use $\emptyset$ to indicate instances that could not be solved within the time and memory limits, with the red dots representing instances for which the reversed task could not be generated.

the greatest impact on the overall time required to solve an instance. Looking at h^{iPDB} (Figures 2f and 2h), the runtime comparison tends to shift in favor of A* and A$_b^*$, since BAE* needs to create two abstraction heuristics for the original task and the reversed task, which can be costly.

Portfolio

To determine and highlight the complementary strength of A*, A$_b^*$, and BAE*, we examine the use of all search strategies as a portfolio (Xu et al. 2008; Ferber and Seipp 2020) by selecting and executing a single planner based on a classification rule. To evaluate the potential of such a portfolio, and thus the usefulness of BAE*, we consider an oracle that selects the best search strategy for each instance so that each instance is solved as fast as possible. Comparing the Single and Oracle columns of Table 3 shows that choosing between A* and BAE* can usually increase the coverage more significantly than choosing between A* and A$_b^*$ (h^{iPDB} is again an exception due to the increased overhead of computing two abstraction heuristics for BAE*). The oracle over A* and BAE* chooses BAE* for about 20% of the instances. Considering an oracle over all three search algorithms, we see that the coverage can increase even further, highlighting the merits of forward, backward, and bidirectional heuristic search in classical planning. We define a simple classifier $\mathcal{C}$ with the following rule. If the number of initial states of the reversed planning task is less than 100 and the time required to generate the reversed planning task is less than one second, then select BAE* (or A$_b^*$), otherwise select A*. This simple classifier $\mathcal{C}$ over A* and BAE* improves coverage for two heuristics, suggesting that in practice it is likely that adding BAE* to a more elaborate portfolio (Sievers et al.

	Single			Simple Classifier $\mathcal{C}$		Oracle		
	A^*	A^*_b	BAE^*	$\{A^*, A^*_b\}$	$\{A^*, BAE^*\}$	$\{A^*, A^*_b\}$	$\{A^*, BAE^*\}$	$\{A^*, A^*_b, BAE^*\}$
$h^{\max}$	847	558	706	762	**856**	848	**878**	**878**
h^{iPDB}	**1009**	732	805	941	986	1026	1022	**1031**
h^{pot}	989	732	841	945	**995**	1007	1019	**1030**

Table 3: Coverage comparison of a portfolio containing A*, A$^*_b$, and BAE* using an oracle selector and a simple classifier $\mathcal{C}$ that attempts to determine whether the planning task is reversible with reasonable resources.

2019) can improve the overall performance.

Discussion and Future Work

BAE* has demonstrated a high level of performance that can outperform A* on single-source single-target shortest path problems (Alcázar, Riddle, and Barley 2020). In classical planning, the picture looks slightly different. The straightforward application of BAE* to classical planning tasks requires a reversed task with an explicit state representation and a potentially exponential size increase. However, our empirical evaluation shows that when it is possible to generate such reversed tasks, BAE* performs comparably well to A* and even outperforms A* in several instances in terms of node expansions. This naturally gives rise to the need for a more appropriate generation of reversed tasks. Therefore, a future research direction is to allow partial states in the backward search (Alcázar et al. 2013), but this would require adjusting BAE* and investigating whether the higher cost of testing for intersections of the search frontiers pays off.

Another interesting future work is to investigate the lower-bound framework of Alcázar, Riddle, and Barley (2020) for classical planning, which combines and generalizes the work of Sadhukhan (2012), i.e., BAE*, with the work of Kaindl and Kainz (1997). However, this also requires a more in-depth understanding of how to handle reversed planning tasks and partial states in backward search.

Finally, in the future, the development of bidirectional heuristic search algorithms for planning would benefit greatly from having a more optimized and robust interface for integration into modern planning systems such as Fast Downward (Helmert 2006). Our current implementation of BAE* can still be improved to be more efficient compared to Fast Downward's highly optimized forward search.

Conclusion

In this paper we have analyzed BAE* for classical planning in theory and practice. For a straightforward application of BAE*, we defined a reversed planning task that does not rely on partial states but has a potentially exponential increase in size. We have alleviated this problem to some extent by using mutexes and reachability analysis, which significantly reduce the number of goal states, making it possible to create such a reversed task in practice for many domains. Our empirical evaluation shows that BAE* can expand significantly fewer nodes compared to forward and backward A* in multiple domains, demonstrating its usefulness. Overall, our analysis shows the merits of forward and backward A* as well as the usefulness of bidirectional heuristic search in the form of BAE* for classical planning.

Acknowledgments

This research was partially supported by TAILOR, a project funded by the EU Horizon 2020 research and innovation programme under grant agreement no. 952215. David Speck was supported by the German Research Foundation (DFG) as part of the EPSDAC project (MA 7790/1-1).

References

Alcázar, V. 2021. 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 *Proc. ICAPS 2021*, 7–15.

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting Regression in Planning. In *Proc. IJCAI 2013*, 2254–2260.

Alcázar, V.; Riddle, P. J.; and Barley, M. 2020. A Unifying View on Individual Bounds and Heuristic Inaccuracies in Bidirectional Search. In *Proc. AAAI 2020*, 2327–2334.

Alcázar, V.; and Torralba, Á. 2015. A Reminder about the Importance of Computing and Exploiting Invariants in Planning. In *Proc. ICAPS 2015*, 2–6.

Bäckström, C.; and Nebel, B. 1995. Complexity Results for SAS$^+$ Planning. *Computational Intelligence*, 11(4): 625–655.

Barker, J. K.; and Korf, R. E. 2015. Limitations of Front-To-End Bidirectional Heuristic Search. In *Proc. AAAI 2015*, 1086–1092.

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

Chen, J.; Holte, R. C.; Zilles, S.; and Sturtevant, N. R. 2017. Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions. In *Proc. IJCAI 2017*, 489–495.

Dijkstra, E. W. 1959. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1: 269–271.

Eckerle, J.; Chen, J.; Sturtevant, N. R.; Zilles, S.; and Holte, R. C. 2017. Sufficient Conditions for Node Expansion in Bidirectional Heuristic Search. In *Proc. ICAPS 2017*, 79–87.

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2015. BDDs Strike Back (in AI Planning). In *Proc. AAAI 2015*, 4320–4321.

Ferber, P.; and Seipp, J. 2020. Explainable Planner Selection. In *ICAPS Workshop on Explainable AI Planning (XAIP)*.

Fišer, D.; Torralba, Á.; and Hoffmann, J. 2022. Operator-Potential Heuristics for Symbolic Search. In *Proc. AAAI 2022*. To appear.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*, volume 7 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool.

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

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proc. AAAI 2007*, 1007–1012.

Helmert, M. 2006. The Fast Downward Planning System. *JAIR*, 26: 191–246.

Holte, R. C.; Felner, A.; Sharon, G.; Sturtevant, N. R.; and Chen, J. 2017. MM: A bidirectional search algorithm that is guaranteed to meet in the middle. *AIJ*, 252: 232–266.

Hu, K.; and Speck, D. 2022. Code and data for the SOCS 2022 paper "On Bidirectional Heuristic Search in Classical Planning: An Analysis of BAE*". https://doi.org/10.5281/zenodo.6570805.

Kaindl, H.; and Kainz, G. 1997. Bidirectional Heuristic Search Reconsidered. *JAIR*, 7: 283–317.

Kissmann, P.; Edelkamp, S.; and Hoffmann, J. 2014. Gamer and Dynamic-Gamer – Symbolic Search at IPC 2014. In *IPC-8 planner abstracts*, 77–84.

Kuroiwa, R.; and Fukunaga, A. 2020. Front-to-Front Heuristic Search for Satisficing Classical Planning. In *Proc. IJCAI 2020*, 4098–4105.

Kwa, J. B. H. 1989. BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm. *AIJ*, 38(1): 95–109.

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

Pohl, I. 1969. Bi-Directional And Heuristic Search In Path Problems. Technical Report 104, Stanford Linear Accelerator Center.

Pohl, I. 1971. Bi-directional search. In Meltzer, B.; and Michie, D., eds., *Machine Intelligence 6*, 127–140. American Elsevier.

Russell, S.; and Norvig, P. 2003. *Artificial Intelligence — A Modern Approach*. Prentice Hall.

Sadhukhan, S. K. 2012. A new Approach to bidirectional heuristic search using error functions. In *1st International Conference on Intelligent Infrastructure at the 47th Annual National Convention Computer Society of India (CSI-2012)*, 239–243.

Sadhukhan, S. K. 2013. Bidirectional Heuristic Search based on Error Estimate. *CSI Journal of Computing*, 2(1–2): S1:57–S1:64.

Seipp, J.; Pommerening, F.; and Helmert, M. 2015. New Optimization Functions for Potential Heuristics. In *Proc. ICAPS 2015*, 193–201.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection. In *Proc. AAAI 2019*, 7715–7723.

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

Speck, D.; Geißer, F.; and Mattmüller, R. 2018. SYMPLE: Symbolic Planning based on EVMDDs. In *IPC-9 planner abstracts*, 91–94.

Speck, D.; Geißer, F.; and Mattmüller, R. 2020. When Perfect Is Not Good Enough: On the Search Behaviour of Symbolic Heuristic Search. In *Proc. ICAPS 2020*, 263–271.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proc. AAAI 2020*, 9967–9974.

Suda, M. 2014. Property Directed Reachability for Automated Planning. *JAIR*, 50: 265–319.

Torralba, Á.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A Symbolic Bidirectional A* Planner. In *IPC-8 planner abstracts*, 105–109.

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

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *JAIR*, 32: 565–606.