Towards Effective Multi-Valued Heuristics for Bi-objective Shortest-Path Algorithms via Differential Heuristics

Han Zhang[1], Oren Salzman[2], Ariel Felner[3], T. K. Satish Kumar[1], Shawn Skyler[3], Carlos Hernández Ulloa[4], Sven Koenig[1]

[1] University of Southern California
[2] Technion - Israel Institute of Technology
[3] Ben-Gurion University
[4] Universidad San Sebastian
zhan645@usc.edu, osalzman@cs.technion.ac.il, felner@bgu.ac.il, tkskwork@gmail.com, shawn@post.bgu.ac.il, carlos.hernandez@uss.cl, skoenig@usc.edu

Abstract

In bi-objective graph search, each edge is annotated with a cost pair, where each cost corresponds to an objective to optimize. We are interested in finding all undominated paths from a given start state to a given goal state (called the Pareto front). Almost all existing works of bi-objective search use single-valued heuristics, which use one number for each objective, to estimate the cost between any given state and the goal state. However, single-valued heuristics cannot reflect the trade-offs between the two costs. On the other hand, multi-valued heuristics use a set of pairs to estimate the Pareto front between any given state and the goal state and are more informed than single-valued heuristics. However, they are rarely studied and have yet to be investigated in explicit state spaces by any existing work. In this paper, we are interested in using multi-valued heuristics to improve bi-objective search algorithms in explicit state spaces. More specifically, we generalize Differential Heuristics (DHs), a class of memory-based heuristics for single-objective search, to bi-objective search, resulting in Bi-objective Differential Heuristics (BO-DHs). We propose several techniques to reduce the memory usage and computational overhead of BO-DHs significantly. Our experimental results show that, with suggested improvement and tuned parameters, BO-DHs can reduce the node expansion and runtime of a bi-objective search algorithm by up to an order of magnitude, paving the way for more effective multi-valued heuristics.

1 Introduction

The task of bi-objective search is to find paths from a given start state to a given goal state in a graph whose edges are annotated with a pair of costs. Each cost corresponds to an objective to optimize, such as travel time, travel distance, and risk. The cost (or, more precisely, the cost pair) of a path is the component-wise sum of the costs of its edges. Bi-objective search is important for many real-world applications, including route planning for power lines considering economic and ecological impacts (Bachmann et al. 2018), inspecting regions of interest with robots considering the cost of motion and coverage (Fu et al. 2019; Fu, Salzman, and Alterovitz 2021), and transporting hazardous materials considering travel distance and risk (Bronfman et al. 2015).

A path π is said to *dominate* a path π' iff π is not worse than π' on any cost and is better than π' on at least one cost. In this paper, we are interested in finding all undominated paths, referred to as the *Pareto-optimal solution set*. Additionally, we refer to the costs of the Pareto solution set as the *Pareto front*.

Several recent works focus on efficiently solving the bi-objective search problem (Hernández et al. 2023; Ren et al. 2022; Ahmadi et al. 2021; Maristany de las Casas et al. 2021; Geißer et al. 2022). Most of them focus on improving dominance checking (Hernández et al. 2023; Ren et al. 2022), that is, the procedure to determine if a node should be pruned, or solving the problem using bi-directional search (Ahmadi et al. 2021; Maristany de las Casas et al. 2021). A rarely-explored and recent direction to improve bi-objective search algorithms is to improve the heuristic guidance (Geißer et al. 2022). Almost all existing works of bi-objective search use *single-valued heuristics* in which the heuristic value for a state is a pair (h_1, h_2), where h_i, $i \in \{1, 2\}$, corresponds to a cost-to-go estimation for the i-th objective. While a single-valued heuristic can estimate the cost-to-go for each objective individually, it does not reflect the trade-off between the two objectives.

Geißer et al. (2022) explore the implications of using a set of pairs as heuristics to estimate the Pareto front between a given state and the goal state and show that these heuristics improve a bi-objective search algorithm in many planning domains. We call such heuristics *multi-valued heuristics*. However, their heuristics only apply to combinatorial domains (where a set of variables describes a state and the graph is implicitly encoded by the available transitions between states), like classical planning. They are not suitable for problems in explicit state spaces (where a coordinate or an index describes a state and the graph is explicitly stored), like path planning on grids or road networks, which are relevant to many real-world applications.

In this paper, we are interested in using multi-valued heuristics to improve bi-objective search algorithms in explicit state spaces, which has not been investigated by any

existing work. More specifically, we generalize Differential Heuristics (DHs) (Goldberg and Werneck 2005; Sturtevant et al. 2009), a class of memory-based heuristics for single-objective search, to bi-objective search. We call the resulting technique Bi-Objective Differential Heuristics (BO-DHs). In single-objective search, given an input graph, the prepossessing algorithm of DHs selects a set of states (called landmarks), computes the minimum path costs from every landmark to every state in the graph, and stores these minimum path costs in a lookup table. During the query phase, a heuristic is computed using the triangle inequality.

The preprocessing algorithm for BO-DHs computes the Pareto fronts from every landmark to every state in the graph and stores them in a lookup table. A multi-value heuristic is computed using a generalized triangle inequality rule in the query phase. We show that this heuristic is consistent (which guarantees that the search algorithm only expands "Pareto-optimal" nodes).

While BO-DH allows fewer node expansions when compared to using a single-valued heuristic, its downsides include the computational overhead for computing the heuristic and the memory usage for storing the lookup tables. We provide an approximation scheme to compress the lookup tables for BO-DHs, significantly reducing both the computational overhead and the memory usage in the preprocessing phase while still guaranteeing that the algorithm finds the entire Pareto front.

In our experiments, we implemented BO-DH with NAMOA*, an existing bi-objective search algorithm, and extensively evaluated it with various parameters on grid and road network graphs. Our experimental results show that, with tuned parameters, BO-DHs can reduce the node expansion and runtime of NAMOA* by up to an order of magnitude. While it is still unclear how to integrate BO-DH with some other widely used techniques, such as dimensionality reduction (Mandow and De La Cruz 2010), in state-of-the-art bi-objective search algorithms, we leave this as a direction for future work.

2 Terminology and Problem Definitions

We use boldface font to denote pairs and p_i, $i \in \{1, 2\}$, to denote the i-th component of a pair $\mathbf{p}$. We define the addition (resp. subtraction) of two pairs $\mathbf{p}$ and $\mathbf{p}'$ as $\mathbf{p} + \mathbf{p}' = (p_1 + p'_1, p_2 + p'_2)$ (resp. $\mathbf{p} - \mathbf{p}' = (p_1 - p'_1, p_2 - p'_2)$). We define the *component-wise maximum* (resp. *component-wise minimum*) of two pairs $\mathbf{p}$ and $\mathbf{p}'$ as $\mathrm{comax}(\mathbf{p}, \mathbf{p}') = (\max(\mathrm{p}_1, \mathrm{p}'_1), \max(\mathrm{p}_2, \mathrm{p}'_2))$ (resp. $\mathrm{comin}(\mathbf{p}, \mathbf{p}') = (\min(\mathrm{p}_1, \mathrm{p}'_1), \min(\mathrm{p}_2, \mathrm{p}'_2))$). We say that $\mathbf{p}$ *weakly dominates* $\mathbf{p}'$, denoted as $\mathbf{p} \preceq \mathbf{p}'$, iff $p_1 \leq p'_1$ and $p_2 \leq p'_2$. We say that $\mathbf{p}$ *(strictly) dominates* $\mathbf{p}'$, denoted as $\mathbf{p} \prec \mathbf{p}'$, iff $\mathbf{p} \preceq \mathbf{p}'$ and $\mathbf{p} \neq \mathbf{p}'$.

Given a set of pairs P, we use $ND(P)$ to denote the subset of P that are not dominated by any other pair in P. Additionally, given a pair $\mathbf{p}$, we define $P - \mathbf{p}$ (resp. $P + \mathbf{p}$) as the set of pairs $\{\mathbf{p}' - \mathbf{p} \mid \mathbf{p}' \in P\}$ (resp. $\{\mathbf{p}' + \mathbf{p} \mid \mathbf{p}' \in P\}$). Slightly abusing the notation, given two sets of pairs P and P', we say that P *weakly dominates* P', denoted as $P \preceq P'$, iff, for any $\mathbf{p}' \in P'$, there exists $\mathbf{p} \in P$ such that $\mathbf{p} \preceq \mathbf{p}'$.

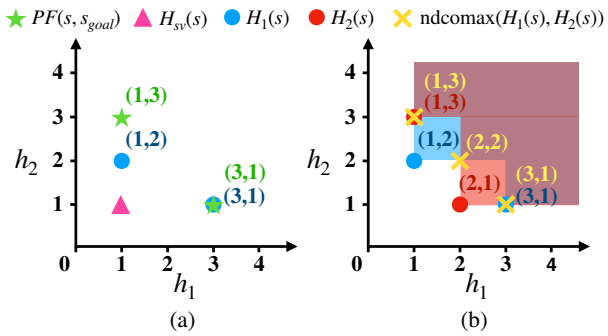


Figure 1: An example of admissible multi-valued heuristic and the ndcomax operation.

A *(bi-objective) graph* is a tuple $G = \langle S, E, \mathbf{c} \rangle$, where S is a finite set of *states*, $E \subseteq S \times S$ is a finite set of *edges*, and $\mathbf{c} : E \to \mathbb{R}^2_{\geq 0}$ is a *cost function* that maps an edge to a positive pair. $succ(s) = \{s' \in S : \langle s, s' \rangle \in E\}$ denotes the successors of state s. In this paper, we focus on undirected graphs. However, all the proposed techniques can be generalized to directed graphs with mild modification. A *path* from state s_1 to state s_ℓ is a sequence of states $\pi = [s_1, s_2 \dots s_\ell]$ with $\langle s_j, s_{j+1} \rangle \in E$ for all $j = 1, 2 \dots \ell - 1$. $\mathbf{c}(\pi) = \sum_{j=1}^{\ell-1} \mathbf{c}(\langle s_j, s_{j+1} \rangle)$ denotes the cost of path π. Path π *dominates (resp. weakly dominates)* a path π' iff $\mathbf{c}(\pi) \prec \mathbf{c}(\pi')$ (resp. $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi')$). Path π is *Pareto-optimal* iff it is not dominated by any other path from s_1 to s_ℓ.

A *query* $q = \langle s_{\text{start}}, s_{\text{goal}} \rangle$ consists of a *start state* s_{start} and a *goal state* s_{goal}. A path is a *solution* to q iff it is from s_{start} to s_{goal}. In this paper, we are interested in finding a *(cost-unique) Pareto-optimal solution set*, that is, a maximal subset of all Pareto-optimal solutions, such that any two solutions in the subset do not have the same cost. For a Pareto-optimal solution set Π, we define the *Pareto front* $P = \{\mathbf{c}(\pi) \mid \pi \in \Pi\}$ as the set of costs of solutions in Π.

We are interested in a problem setting that consists of the preprocessing and query phases. During the *preprocessing phase*, only the input graph G is revealed to the algorithm. The algorithm is allowed to process G and build auxiliary data in the memory for future use. During the *query phase*, queries on G are given to the algorithm, and the algorithm needs to compute the Pareto-optimal solution sets for them. This setting is similar to many real-world applications, where the search algorithm needs to solve multiple problem instances on the same graph. Our primary focus is to improve the query phase runtime.

3 Algorithmic Background

In this section, we review multi-valued heuristics, bi-objective search algorithms, and (single-objective) DHs.

3.1 Multi-Valued Heuristics

A *multi-valued heuristic* H in bi-objective search is a function that maps a state to a set of pairs, that is, $\forall s \in S, H(s) \subset \mathbb{R}^2$. Single-valued heuristics can be viewed as a special case of multi-valued heuristics whose returned values are always

of size one. Heuristic value $H(s)$ estimates the Pareto front between state s and s_{goal}. A multi-valued heuristic is said to be admissible iff it always "lower-bounds" the Pareto front of the cost-to-go. Formally:

Definition 1. *Given a goal state s_{goal}, let $PF(s, s_{\text{goal}})$ be the Pareto front between state s and s_{goal}. A heuristic H is admissible iff, for any state $s \in S$, $H(s) \preceq PF(s, s_{\text{goal}})$.*

Example 1. *Fig. 1(a) demonstrates an admissible bi-objective heuristic. The green stars represent a Pareto front between a state s and state s_{goal} with $PF(s, s_{\text{goal}}) = \{(1,3),(3,1)\}$. The pink triangle represents the best possible (admissible) single-valued heuristic for s $H_{sv}(s) = \{(1,1)\}$. The blue dots represent an admissible multi-valued heuristic $H_1(s) = \{(1,2),(3,1)\}$. It is easily verifiable that $H_1(s) \preceq PF(s, s_{\text{goal}})$. $H_1(s)$ shows that, for any path π between s and s_{goal}, we must have $c_2(\pi) >= 2$ in order to achieve $c_1(\pi) < 3$ and $c_1(\pi) >= 3$ in order to achieve $c_2(\pi) < 2$. However, a single-valued heuristic cannot reflect such information. Intuitively, H_1 is more informed than H_{sv} in estimating the cost-to-go from state s.*

Definition 2. *A heuristic H is consistent iff (i) $\mathbf{0} \in H(s_{\text{goal}})$ and, (ii) for every two states s and s' and every path π from s to s', $H(s) \preceq H(s') + \mathbf{c}(\pi)$.*

Similar to heuristics in single-objective search, a multi-value heuristic is admissible if it is consistent (but not vice versa). Note that, by definition, removing dominated members from heuristic values will not affect the admissibility or consistency. Therefore, we only consider heuristic values with dominated members removed, that is, $\forall s \in S$, $H(s) = ND(H(s))$.

In single-objective search, the maximization between two consistent (or admissible) heuristics results in a new consistent (or resp. admissible) heuristic. Geißer et al. (2022) show that, in bi-objective search, the maximization over two heuristics can be done by the *comax operation between two sets* (defined differently from the comax between two pairs), and comax also preserves the admissibility and consistency. Given two sets of pairs P' and P'', their *comax* is defined as

$$\text{comax}(P', P'') = \{\text{comax}(\mathbf{p}', \mathbf{p}'') \mid \mathbf{p}' \in \text{P}', \mathbf{p}'' \in \text{P}''\}.$$

Additionally, we apply ND to remove the dominated members after computing the comax and define the *ndcomax* of P' and P'' as

$$\text{ndcomax}(P', P'') = ND(\text{comax}(P', P'')).$$

The following example shows the geometric interpretation behind the ndcomax operation:

Example 2. *We add a new admissible heuristic H_2 to Ex. 1. Specifically, $H_2(s) = \{(1,3),(2,1)\}$ and is depicted by the red dots in Fig. 1(b). The yellow crosses visualize $\text{ndcomax}(H_1(s), H_2(s))$ where*

$$\begin{aligned}\text{ndcomax}(H_1(s), H_2(s)) &= ND(\{(1,3),(2,2),(3,1),(3,3)\}) \\ &= \{(1,3),(2,2),(3,1)\}.\end{aligned}$$

It is easy to verify that both H_2 and $\text{ndcomax}(H_1(s), H_2(s))$ weakly dominate $PF(s, s_{\text{goal}})$. Now, given a set of pairs P, we define its dominated region *as the set of all pairs that are weakly dominated by at least one pair in P. From Def. 1, if a heuristic H is admissible, $PF(s, s_{\text{goal}})$ must completely lie in (i.e., be a subset of) the dominated region of $H(s)$. The blue and red regions in Fig. 1(b) represent the dominated regions of $H_1(s)$ and $H_2(s)$, respectively. We can see that the dominated region of $\text{ndcomax}(H_1(s), H_2(s))$ is exactly the intersection of the dominated regions for $H_1(s)$ and $H_2(s)$.*

Algorithm 1: BO-BFS

```
   Input : G = ⟨S, E⟩, s_start, s_goal, H
 1 n_root ← new node at s_start with g(n_root) = (0, 0) and
     parent(n_root) = None
 2 initialize Open and add n_root to it
 3 Solutions ← ∅
 4 while Open ≠ ∅ do
 5     n ← Open.pop()
 6     if is_dominated_1(n) then continue
 7     if s(n) = s_goal then
 8         add n to Solutions
 9         continue
10     for s' ∈ succ(s(n)) do
11         for h ∈ H(s') do
12             n' ← new node at s' with
                 g(n') = g(n) + c((s, s')), h(n') = h, and
                 parent(n') = n
13             if is_dominated_2(n') then continue
14             Open.insert(n')
15 return Solutions
```

3.2 Bi-Objective Search Algorithms

Many bi-objective search algorithms, such as NAMOA* (Mandow and De La Cruz 2010), NAMOA*dr (Pulido, Mandow, and De la Cruz 2015), and BOA* (Hernández et al. 2023), conform to the same algorithmic framework. We call this framework Bi-Objective Best-First Search (BO-BFS). BO-BFS is similar to A*, but, most differently, it needs to consider multiple nodes (with costs that do not weakly dominate each other) for the same state.

Alg. 1 shows the Pseudo-code for BO-BFS. BO-BFS maintains a priority queue $Open$, which contains the generated but not-yet-expanded nodes. Each node n contains a state $s(n)$, a $\mathbf{g}$-value $\mathbf{g}(n)$, an $\mathbf{h}$-value $\mathbf{h}(n)$, and an $\mathbf{f}$-value $\mathbf{f}(n) = \mathbf{g}(n) + \mathbf{h}(n)$.

In each iteration, BO-BFS extracts a node n with the lexicographically smallest $\mathbf{f}$-value from $Open$ (Line 5). BO-BFS then performs *dominance checks* for node n to determine if n or n's child nodes have the potential to be included in $Solutions$ and discards n if it does not pass the check (Line 6). If n is not pruned, the algorithm either adds it to the solution set if $s(n) = s_{\text{goal}}$ (Lines 7-9) or expands it (Lines 10-14) otherwise. When expanding a node n, BO-BFS generates a child node for each successor s' of $s(n)$ and each pair $\mathbf{h}$ in $H(s')$. BO-BFS also performs *dominance checks* for each child node when generating it (Line 13). Note that the dominance check procedure for generated nodes can be different from the one for nodes

extracted from $Open$. When $Open$ becomes empty, the algorithm terminates and returns the solution set. Different BO-BFS algorithms differ in how functions is_dominated_1, is_dominated_2, and insert_to_open are implemented.

In this paper, we focus on NAMOA* because, different from NAMOA*dr and BOA*, it is directly compatible with multi-valued heuristics. When generating a node n', function is_dominated_2 of NAMOA* checks (i) if $\mathbf{g}(n')$ is dominated by the $\mathbf{g}$-value of any generated node with state $s(n')$ and (ii) if $\mathbf{f}(n')$ is weakly dominated by the $\mathbf{g}$-value of any node in $Solutions$. When inserting n' to $Open$, NAMOA* removes nodes with state $s(n')$ whose $\mathbf{g}$-values are weakly dominated by $\mathbf{g}(n')$ from $Open$. When NAMOA* extracts a node n from $Open$, it guarantees that $\mathbf{g}(n)$ is not dominated by the $\mathbf{g}$-value of any generated nodes with state $s(n)$. Therefore, function is_dominated_1 only checks if $\mathbf{f}(n)$ is weakly dominated by the $\mathbf{f}$-value of any solution node and $\mathbf{g}(n)$ is weakly dominated by the $\mathbf{g}$-value of any expanded node with state $s(n)$.

Example 3. *Fig. 2 shows an example bi-objective graph. There are two Pareto-optimal paths from s_{start} to s_{goal}, shown in Fig. 2(b). We now consider the behavior of NAMOA* with the single-valued heuristic shown in Fig. 2(a). Note that this heuristic is the best-possible single-valued heuristic since each component of a heuristic value is exactly the minimum cost-to-go for the corresponding objective. We use triple $(s(n), \mathbf{g}(n), \mathbf{f}(n))$ to refer to a node n.*

1. *In Iteration 1, NAMOA* generates nodes $n_1 = (s_1, (1,1), (3,5))$, $n_2 = (s_2, (1,1), (4,4))$, and $n_3 = (s_3, (1,1), (5,3))$.*
2. *In Iteration 2, NAMOA* extracts n_1 (from Open) and generates node $n_4 = (s_{\text{goal}}, (3,7), (3,7))$. Note that the child nodes at states s_{start} and s_2 are pruned by dominance checks because of n_{root} and n_2, respectively.*
3. *In Iteration 3, NAMOA* extracts n_4 and finds solution π_1.*
4. *In Iteration 4, NAMOA* extracts n_2 and generates node $n_5 = (s_4, (2,2), (6,6))$.*
5. *In Iteration 5, NAMOA* extracts n_3 and generates node $n_6 = (s_{\text{goal}}, (7,3), (7,3))$.*
6. *In Iteration 6, NAMOA* extracts n_5 and does not generate any node.*
7. *In Iteration 7, NAMOA* extracts n_6 and finds solution π_2.*
8. *As Open becomes empty, NAMOA* terminates and return the two Pareto-optimal solutions it has found.*

Note that, while no Pareto-optimal solution via s_2 or s_4 exists, NAMOA still expands nodes n_2 and n_5.*

Neither NAMOA*dr nor BOA* can currently work with multi-valued heuristics because they use a technique called *dimensionality reduction* to improve the efficiency of dominance checks. Dimensionality reduction exploits that nodes are extracted in lexicographically increasing $\mathbf{f}$-values (and hence also lexicographically increasing $\mathbf{g}$-values for a specific state when the algorithms use a single-value heuristic). To compare a node n against all expanded nodes with state $s(n)$, NAMOA*dr and BOA* do not need to compare g_1-values and hence only check the max g_2-value of all these expanded nodes, which can be done in constant time. Combining dimensionality reduction and multi-valued heuristics remains an interesting direction for future work.

Notably, other existing algorithms like BDijkstra (Sedeño-Noda and Colebrook 2019) and BOD (Hernández et al. 2023) share a similar structure as BO-BFS but generalize Dijkstra's algorithm instead of A* to bi-objective search. These algorithms find the Pareto-optimal solution sets between a given state s and all other states.

3.3 DHs

A (single-objective) DH is a consistent memory-based heuristic using a pre-computed lookup table. In the preprocessing phase, a set of landmark states $L \subset S$ are selected. Then, for each landmark state $\ell \in L$, the preprocessing algorithm computes the minimum path cost between ℓ and every other state and stores these costs in the lookup table.

During the query phase, the DH value for a state is evaluated on-demand using the lookup table: Let $d(s, s'), s \in S, s' \in S$, denote the minimum path cost between states s and s'. For each landmark $\ell \in L$, a given goal state s_{goal}, and any state $s \in S$, we have that $d(\ell, s_{\text{goal}}) \leq d(\ell, s) + d(s, s_{\text{goal}})$ and $d(\ell, s) \leq d(s, s_{\text{goal}}) + d(\ell, s_{\text{goal}})$ from the triangle inequality. Consequently, the DH is computed as

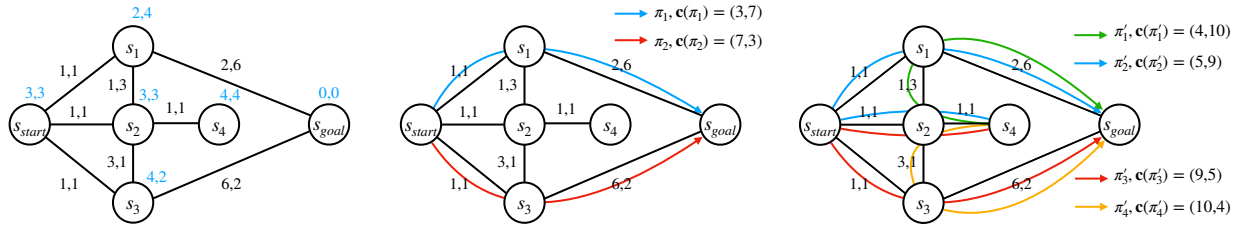
$$h_{\text{DH}}(s) = \max_{\ell \in L} \{\max(d(\ell, s_{\text{goal}}) - d(\ell, s), \\ d(\ell, s) - d(\ell, s_{\text{goal}}))\}.$$

Example 4. *Consider the single-objective graph obtained from Fig. 2(a) by only considering the first cost and assume that state s_4 is the only selected landmark. It is easy to verify that $h_{\text{DH}}(s) = max(d(s, s_4) - d(s_{\text{goal}}, s_4), d(s_4, s) - d(s_4, s_{\text{goal}}))$ is admissible. For example, we have $d(s_{\text{start}}, s_4) = 2$ (following path $[s_{\text{start}}, s_2, s_4]$) and $d(s_{\text{goal}}, s_4) = 4$ (following path $[s_{\text{goal}}, s_1, s_2, s_4]$). Therefore, we have $h_{\text{DH}}(s_{\text{start}}) = max(4-2, 2-4) = 2$, which is not larger than $d(s_{\text{start}}, s_{\text{goal}}) = 3$ (following path $[s_{\text{start}}, s_1, s_{\text{goal}}]$).*

Intuitively, a "good" landmark selection should well cover the input graph. We focus on the so-called *Avoid* strategy, first proposed by Goldberg and Werneck (2005) and has been used in many existing works (Goldberg, Kaplan, and Werneck 2006; Goldenberg et al. 2011). Roughly speaking, this strategy iteratively selects a new landmark that "avoids" the states "covered" by already selected landmarks until a given number of landmarks are selected.

4 BO-DHs

In this section, we first describe BO-DHs, which generalize DHs to bi-objective search. A BO-DH uses a lookup table to store the Pareto front between every landmark and every state, which can incur large memory usage. The computational overhead for a BO-DH can also be significant. To address these limitations, we then describe an approach to compress the lookup table, which reduces memory usage and computational overhead while retaining admissibility.



(a) (b) (c)

Figure 2: An example bi-objective search instance. (a) Single-valued heuristic (blue numbers). (b) Pareto-optimal solutions from s_{start} to s_{goal}. (c) Pareto-optimal solutions from s_4 to s_{goal}.

4.1 Basic BO-DHs

Given a graph G and a set of landmarks L, let $PF(\ell, s)$ denote the Pareto front between a landmark $\ell \in L$ and any state s. We show the following lemma:

Lemma 1. *For any $\mathbf{p}' \in PF(\ell, s_{\text{goal}})$ and any $s \in S$, the set $PF(\ell, s) - \mathbf{p}'$ weakly dominates $PF(s, s_{\text{goal}})$.*

Proof. Assume that $PF(\ell, s) - \mathbf{p}'$ does not weakly dominate $PF(s, s_{\text{goal}})$. Thus, there exists some $\mathbf{p} \in PF(s, s_{\text{goal}})$ such that $\mathbf{p} + \mathbf{p}'$ is not dominated by any pair in $PF(\ell, s)$. Recall that $\mathbf{p}$ and $\mathbf{p}'$ are the costs of two paths from s to s_{goal} and from ℓ to s_{goal}, respectively. By connecting these two paths, we obtain a path from s to ℓ with a cost not weakly dominated by $PF(\ell, s)$, which contradicts that $PF(\ell, s)$ is a Pareto front. □

Let P and P' be two sets of pairs and denote

$$P - P' := \underset{\mathbf{p}' \in P'}{\text{ndcomax}}\{P - \mathbf{p}'\}.$$

For each landmark ℓ, we define the following heuristics[1]:

$$H_f^\ell(s) := PF(\ell, s) - PF(\ell, s_{\text{goal}}) \text{ and}$$
$$H_b^\ell(s) := PF(\ell, s_{\text{goal}}) - PF(\ell, s).$$

Heuristic $H_f^\ell(s)$ is the ndcomax of $PF(\ell, s) - \mathbf{p}'$ for all $\mathbf{p}' \in PF(\ell, s_{\text{goal}})$, which is admissible because ndcomax preserves admissibility. As the graph is undirected, we can derive a similar heuristic from the opposite direction, resulting in another admissible heuristic $H_b^\ell(s)$. Finally, we derive the BO-DH by taking the ndcomax over all landmarks ℓ as

$$H_{\text{BO-DH}}(s) = \underset{\ell \in L}{\text{ndcomax}}\left\{\text{ndcomax}\left(H_f^\ell(s), H_b^\ell(s)\right)\right\}. \quad (1)$$

Example 5. *Assume that state s_4 in Fig. 2 is the only landmark selected and consider the BO-DH value for state s_2. From Fig. 2(c), we have $PF(s_4, s_{\text{goal}}) = \{(4,10),(5,9),(9,5),(10,4)\}$. It is easy to see that $PF(s_4, s_2) = \{(1,1)\}$. Therefore, we have that $H_b^{s_4}(s_2) = PF(s_4, s_{\text{goal}}) - PF(s_4, s_2) = \{(3,9),(4,8),(8,4),(9,3)\}$ and that $H_f^{s_4}(s_2) = PF(s_4, s_2) - PF(s_4, s_{\text{goal}}) = \text{ndcomax}\{\{(-3,-9)\},\{(-4,-8)\},\{(-8,-4)\},\{(-9,-3)\}\} = \{(-3,-3)\}$. We have that $H_{\text{BO-DH}}(s_2) = \text{ndcomax}(H_f^{s_4}(s_2), H_b^{s_4}(s_2)) = \{(3,9),(4,8),(8,4),(9,3)\}$. If we run NAMOA* with $H_{\text{BO-DH}}$, NAMOA* will generate four nodes at s_2, with $\mathbf{f}$-values $(4,10),(5,9),(9,5)$, and $(10,4)$, respectively, in the first iteration. None of these nodes will be expanded because they are weakly dominated by either $\mathbf{c}(\pi_1)$ or $\mathbf{c}(\pi_2)$.*

Theorem 1. *$H_{\text{BO-DH}}(s)$ is consistent.*

We omit the proof due to the page limit.

4.2 Compressed BO-DHs

In practice, storing the entire Pareto front between every landmark and every state can require large memory. Additionally, evaluating Eq. 1 can be time-consuming since it requires computing the ndcomax of multiple sets of pairs. However, one might consider using a smaller set of pairs to "approximate" the entire Pareto front.

Let $\ell \in S$ be a landmark and recall that $PF(\ell, s_{\text{goal}})$ denotes the Pareto front from ℓ to s_{goal}. We propose an approach to compress the lookup table of a BO-CH based on the following two observations:

O1 Let $\overline{PF}(\ell, s_{\text{goal}})$ be a set of pairs such that $PF(\ell, s_{\text{goal}}) \preceq \overline{PF}(\ell, s_{\text{goal}})$, i.e., for any $\mathbf{p} \in \overline{PF}(\ell, s_{\text{goal}})$, there exists $\mathbf{p}' \in PF(\ell, s_{\text{goal}})$ such that $\mathbf{p}' \preceq \mathbf{p}$. Following Lemma 1, we have that

$$\forall s \in S, PF(\ell, s) - \mathbf{p} \preceq PF(\ell, s) - \mathbf{p}' \preceq PF(s, s_{\text{goal}}).$$

Therefore, we have that $PF(\ell, s) - \overline{PF}(\ell, s_{\text{goal}})$ weakly dominates $PF(s, s_{\text{goal}})$.

O2 Let $\underline{PF}(\ell, s)$ be a set of pairs such that $\underline{PF}(\ell, s) \preceq PF(\ell, s)$ for some $s \in S$. Then $\underline{PF}(\ell, s) - \overline{PF}(\ell, s_{\text{goal}})$ weakly dominates $PF(s, s_{\text{goal}})$, too.

Conceptually, we can view the weakly dominance relation $\preceq$ as a bi-objective generalization of the less-than-or-equal-to relation $\leq$. We have that

$$\forall s \in S\ \underline{PF}(\ell, s) - \overline{PF}(\ell, s_{\text{goal}}) \preceq PF(\ell, s) - PF(\ell, s_{\text{goal}}) \preceq PF(s, s_{\text{goal}})$$

[1]Here, 'f' and 'b' refer to forward and backward, respectively.

Algorithm 2: Compressing Pareto fronts

```
   Input  : P,ε
1  [p_1, p_2 ... p_N] ← P sorted in the lexicographically
     increasing order
2  L ← [⟨p_1, p_1⟩]
3  foreach p ∈ [p_2, p_3 ... p_N] do
4  |  AP ← the last element in L
5  |  AP_new ← merge(AP, ⟨p, p⟩)
6  |  if AP_new is ε-bounded then
7  |  |  remove AP from L
8  |  |  insert AP_new into L
9  |  else insert ⟨p, p⟩ into L
10 P ← {a | ⟨a, p⟩ ∈ L}
11 P̄ ← {p | ⟨a, p⟩ ∈ L}
12 return P, P̄
13 Function merge(AP = ⟨a, p⟩, AP' = ⟨a', p'⟩):
14 |  a_new ← comin(a, a')
15 |  if p' ≤_ε a_new then p_new ← p'
16 |  else p_new ← p
17 |  return ⟨a_new, p_new⟩
```

because $PF(\ell, s_{\text{goal}}) \preceq \overline{PF}(\ell, s_{\text{goal}})$ and $\underline{PF}(\ell, s) \preceq PF(\ell, s)$. Thus, we can use $\underline{PF}$ and $\overline{PF}$, whose sizes can be smaller than the sizes of Pareto fronts, for heuristic computation and hence reduce the memory requirement for BO-DHs.

We modify the merging procedure of A*pex (Zhang et al. 2022), a state-of-the-art approximate bi-objective search algorithm, and use it to compute $\underline{PF}$ and $\overline{PF}$. The resulting algorithm takes a parameter $\varepsilon \geq 0$ as input, and, as we will see, the sizes of the computed $\underline{PF}$ and $\overline{PF}$ decrease as ε increases. Therefore, we can use ε to control the sizes of $\underline{PF}$ and $\overline{PF}$. Before we describe the algorithm, we review the notion of *apex-path tuples*, introduced by Zhang et al. (2022): An apex-path tuple $AP = \langle \mathbf{a}, \mathbf{p} \rangle$ is a tuple of two pairs, representing a set of pairs P, with $\mathbf{a}$, called the *apex*, being the component-wise minimum of all pairs in P, and $\mathbf{p} \in P$, called the *representative pair*. An apex-path tuple AP is said to be ε-bounded iff $\mathbf{p} \preceq_\varepsilon \mathbf{a}$ (i.e., $p_1 \leq (1+\varepsilon) \cdot a_1$ and $p_2 \leq (1+\varepsilon) \cdot a_2$). We define *merging* two apex-path tuples as the operation that creates a new apex-path tuple whose apex is the component-wise minimum of the two input apexes, and the new representative pair is selected from the input representative pairs.

We are now ready to describe our approach to compute $\underline{PF}$ and $\overline{PF}$ from a given Pareto front, detailed in Alg. 2. The input to Alg. 2 is a set of pairs P and an $\varepsilon \geq 0$. Alg. 2 outputs two sets of pairs $\underline{P}$ and $\overline{P}$ with $\underline{P} \preceq P \preceq \overline{P}$. Let $[\mathbf{p}_1, \mathbf{p}_2 \dots \mathbf{p}_N]$ denote the sequence of pairs in P sorted in the lexicographically increasing order. Alg. 2 begins with a list of apex-path tuples $L = [\langle \mathbf{p}_1, \mathbf{p}_1 \rangle]$ and iterates over $[\mathbf{p}_2 \dots \mathbf{p}_N]$ (Lines 3-9). For each $\mathbf{p} \in [\mathbf{p}_2 \dots \mathbf{p}_N]$, Alg. 2 tries to merge apex-path tuple $\langle \mathbf{p}, \mathbf{p} \rangle$ with the last element in L if the resulting apex-path tuple is ε-bounded or inserts $\langle \mathbf{p}, \mathbf{p} \rangle$ to L otherwise. After iterating over all pairs, Alg. 2 returns $\underline{P}$ as the set of apexes in L and $\overline{P}$ as the set of representative pairs in L (Lines 10-12). We have that $\underline{P} \preceq P$ because, for each pair $\mathbf{p} \in P$, there exists a pair $\mathbf{p}' \in \underline{P}$ that is equal to $\mathbf{p}$ or is the comin of $\mathbf{p}$ and other pairs. We have that $\mathbf{p}' \preceq \mathbf{p}$ in either case. We have that $P \preceq \overline{P}$ because $\overline{P}$ is a subset of P. Moreover, when a larger ε is used, Alg. 2 is more likely to merge apex-path tuples and hence outputs $\underline{P}$ and $\overline{P}$ with smaller sizes.

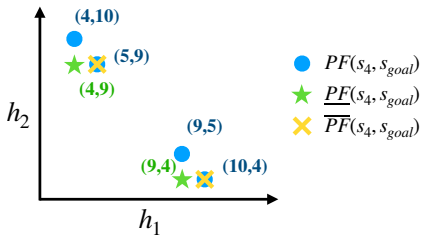


Figure 3: An example of compressing $PF(s_4, s_{\text{goal}})$ from Ex. 5 with $\varepsilon = 0.3$.

Example 6. *Continue Ex. 5 and assume that Alg. 2 receives* $PF(s_4, s_{\text{goal}}) = \{(4,10), (5,9), (9,5), (10,4)\}$ *and* $\varepsilon = 0.3$ *as the input. The output of Alg. 2 is* $\underline{PF}(s_4, s_{\text{goal}}) = \{(4,9), (9,4)\}$ *and* $\overline{PF}(s_4, s_{\text{goal}}) = \{(5,9), (10,4)\}$ *(see Fig. 3). Note that* $(4,10)$ *and* $(5,9)$ *are merged since* $(5,9) \preceq_{0.3} (4,9)$. *However, the resulting apex-path tuple is not further merged with* $(9,5)$ *since neither* $(5,9)$ *or* $(9,5)$ 0.3*-dominates the new apex* $(4,5)$.

Now, we propose BO-DH(ε)s, our compressed variant of BO-DHs. For each landmark ℓ and every state s, instead of storing $PF(\ell, s)$, the lookup table stores only $\underline{PF}(\ell, s)$ and $\overline{PF}(\ell, s)$ as computed by Alg. 2. The BO-DH(ε) is then computed as

$$H_{\text{BO-DH}(\varepsilon)}(s) = \underset{\ell \in L}{\text{ndcomax}} \left\{ \text{ndcomax} \left(H^{\ell}_{f,\varepsilon}(s), H^{\ell}_{b,\varepsilon}(s) \right) \right\}, \tag{2}$$

with

$$H^{\ell}_{f,\varepsilon}(s) = \underline{PF}(\ell, s) - \overline{PF}(\ell, s_{\text{goal}}) \text{ and}$$
$$H^{\ell}_{b,\varepsilon}(s) = \underline{PF}(\ell, s_{\text{goal}}) - \overline{PF}(\ell, s).$$

Example 7. *Since* $PF(s_4, s_2)$ *contains only one pair* $(1,1)$, $\overline{PF}(s_4, s_2)$ *is the same as* $PF(s_4, s_2)$. *We omit the computation of* $H^{s_2}_{f,0.3}(s_4)$ *since it only contains negative pairs. Using Eq. 2, we have* $H_{\text{BO-DH}(0.3)}(s_4) = \underline{PF}(s_4, s_{\text{goal}}) - \overline{PF}(s_4, s_2) = \{(3,8), (8,3)\}$. *If we run NAMOA* with this compressed BO-DH, when expanding the root node, NAMOA* will generate only two nodes for state* s_2, *with* $\mathbf{f}$*-values* $(4,9)$ *and* $(9,4)$. *These two nodes will not be eventually expanded either. Note that NAMOA* generates fewer nodes here by using the compressed BO-DH.*

5 Combining BO-DHs and BO-BFS

In this section, we describe our approach to combining BO-DHs and BO-BFS. In the preprocessing phase, the algorithm chooses the set of landmarks L using the Avoid strategy and uses BOD (Hernández et al. 2023) to compute the

single-to-all Pareto fronts for each landmark. It also compresses the lookup tables for each landmark if a positive ε is given.

In the query phase, we use both the BO-DH and the *best possible single-valued heuristic* (computed by running Dijkstra's algorithm from the goal state for each objective individually) by taking the ndcomax of them. The BO-DH for a state is unknown initially and evaluated only when the algorithm generates nodes with this state (Lines 11-14 of Alg. 1). We also cache the heuristic for a state and reuse it for every node with this state. Our preliminary study shows that a naive implementation of BO-DHs incurred large computational overhead and was unpractical. We hence propose the following speedup techniques.

Active landmarks: We implement a strategy that dynamically activates "useful" landmarks during the search and only uses active landmarks for evaluating the BO-DH. A similar strategy has also been used in single-objective DHs (Goldberg and Werneck 2005).

We first describe how we measure the "usefulness" of a multi-value heuristic to the given query. Consider a state s and let $h_1^{\min}$ and $h_2^{\min}$ denote the minimum c_1 and c_2 costs between s and s_{goal}, respectively. Given a multi-valued heuristic H, we define $A_s(H)$ as the area of the region weakly dominated by pair $(h_1^{\min}, h_2^{\min})$ but not $H(s)$. Finally, we define $Q_s(H) := 1+A_s(H)/(h_1^{\min} \cdot h_2^{\min})$, where we normalize $A_s(H)$ with $(h_1^{\min} \cdot h_2^{\min})$. Conceptually, a larger value of $Q_s(H)$ (or $A_s(H)$) means that heuristic H indicates a larger trade-off that needs to be made for the two objectives (and hence H is more informed). Please refer to the appendix for the intuition behind Q_s.

We now detail our proposed strategy for activating landmarks, parameterized by an *update interval* i and an *update threshold* δ. The search algorithm begins with an empty set of active landmarks and attempts to update the active landmark set when evaluating the heuristics for every i-th state s (including the start state). When attempting to update the set of active landmarks, the algorithm finds the inactive landmark that improves the Q_s-value the most and activates this landmark if the new Q_s-value is at least $1 + \delta$ better than the current Q_s-value. It repeats this process of adding landmarks until no inactive landmark can improve the heuristic value by more than the given threshold.

Speeding up the ndcomax operation: According to its definition, Computing the ndcomax of two sets P_1 and P_2 needs to iterate their cross product. When evaluating BO-DHs, the ndcomax operations are performed multiple times and can be very time-consuming.

However, as Ex. 2 shows, computing the ndcomax is equivalent to computing the intersection of the regions that P_1 and P_2 individually weakly dominate. Given several sets of pairs $P_1, P_2 \dots P_n$, let N denote the total numbers of pairs in these sets $N = \sum_i |P_i|$. We sketch our approach to compute $P_{result} = \text{ndcomax}\{P_1, P_2 \dots P_n\}$ in $O(N \log(N))$ time: For a set of pairs P and the sequence of its pairs sorted lexicographically $\mathbf{p^{(1)}}, \mathbf{p^{(2)}} \dots \mathbf{p^{(m)}}$, we denote $\{(p_1^{(2)}, p_2^{(1)}), (p_1^{(3)}, p_2^{(2)}) \dots (p_1^{(m)}, p_2^{(m-1)})\}$ as the *corner points* of P. A set of pairs can be described by (i) its corner points and (ii) the minimum values of the first and second components. Following the geometric intuition of ndcomax, any corner point of P_{result} is also a corner point of an input set P_i that does not dominate any other corner points. Therefore, computing the corner points of P_{result} is equivalent to computing its "non-dominating" subset. To do so, we sort the corner points of $P_1, P_2 \dots P_n$ in the lexicographically decreasing order (in $N \log(N)$ time) and iterate over them. A corner point does not dominate any other corner point (and hence is a corner point of P_{result}) iff its p_2-value is larger than the maximum p_2-value so far, which can be checked in constant time.

Our preliminary results indicate that the naive ndcomax implementation is slower than the improved ndcomax implementation by about two orders of magnitude.

6 Experimental Results

In this section, we evaluate BO-DHs on two grids and two road networks. The two grids are the 32×32 empty grid and the den520d grid (257×256) from an existing benchmark.[2] For these two grids, we randomly sample each component of the edge cost pair from the integers within $[1, 10]$, which follows the convention in Pulido, Mandow, and De la Cruz (2015) and Ren et al. (2022). The two road networks are the NY ($264,346$ states and $733,846$ edges) and BAY ($321,270$ states and $794,830$ edges) maps from the 9th DIMACS Implementation Challenge: Shortest Path.[3] We use the travel distance and time from the DIMACS data set as the edge costs. For each graph, we use 100 randomly generated queries.

We use NAMOA* with the best possible single-valued heuristics as our baseline algorithm and compare it to NAMOA* with BO-DHs (denoted as NAMOA*+BO-DH). Our implementation also follows the lazy dominance check approach of BOA* and skips the time-consuming iteration of $Open$. We implemented all algorithms in C++[4] and ran all experiments on a MacBook Pro with an M1 Pro CPU and 32GB of memory.

Comparing different parameters for BO-DH: We start by evaluating the impact of different parameters on memory usage (of the lookup table) and query time. Here, we focus on the den520d grid. Recall that NAMOA*+BO-DH has four parameters: number of landmarks $|L|$, approximation factor ε, updating interval i, and updating threshold δ.

To study the impact of the approximation factor on memory usage and query runtime, we evaluated four ε-values 0, 0.001, 0.005, and 0.01. We use $L = 128$ landmarks, $\delta = 0.001$, and $i = \infty$ (the algorithm attempts to activate landmarks only when generating the root search node). Results, summarized in Table 1, show that a larger approximation factor significantly decreases the memory usage of the lookup table. Note that the memory usage of BO-DH

[2] https://movingai.com/benchmarks/grids.html

[3] http://users.diag.uniroma1.it/challenge9/download.shtml.

[4] https://github.com/HanZhang39/BODifferentialHeuristics

Algorithm	Memory	t_{query}	$t_{eval\text{-}h}$	#exp
NAMOA*	0	1.13	0	430K
$\varepsilon = 0$	7.2GB	1.84	1.38	57K
$\varepsilon = 0.001$	10.4GB	1.26	0.82	58K
$\varepsilon = 0.005$	2.5GB	0.62	0.12	73K
$\varepsilon = 0.01$	1.3GB	0.73	0.04	99K

Table 1: Experimental results for NAMOA*+BO-DH with different ε-values on the den520d grid. For each variant of NAMOA*+BO-DH, we report the memory usage for the look table, the average runtime for solving an instance (t_{query}, in seconds), the average runtime for evaluating heuristics ($t_{eval\text{-}h}$, in seconds), and the average number of expanded nodes (#exp).

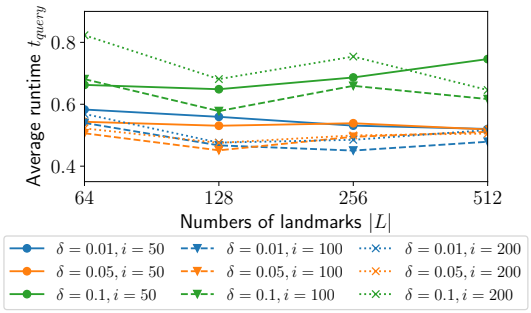


Figure 4: Average runtime for NAMOA*+BO-DH with different parameters over the 100 instances on den520d. The ε-value is 0.01. The average runtime for NAMOA* is 1.13s.

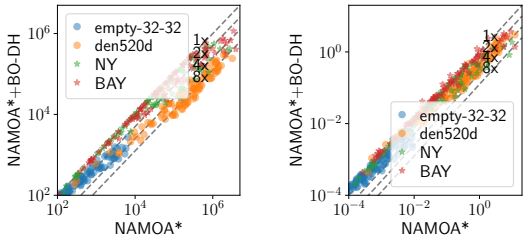
with $\varepsilon = 0.001$ is larger than that of BO-DH with $\varepsilon = 0$ because its lookup table needs to store both $\underline{PF}$ and $\overline{PF}$ while the lookup table for BO-DH with $\varepsilon = 0$ only stores PF. With a smaller ε-values, the search algorithm expands fewer nodes on average. However, the search algorithms with $\varepsilon = 0$ and $\varepsilon = 0.001$ has larger average runtime than those with $\varepsilon = 0.005$ and $\varepsilon = 0.01$ because evaluating heuristics incurred larger computational overhead.

We also evaluated NAMOA*+BO-DH with different numbers of landmarks $|L|$, updating thresholds δ, and updating intervals i on den520d with an ε-value of 0.01. Fig. 4 shows the average runtime of search algorithms for different numbers of landmarks. The average runtime of NAMOA* (1.13s) is larger than the runtimes of all NAMOA*+BO-DH variants. Each line represents a combination of δ and i. Different colors (blue, yellow, and green) represent different δ values of 0.01, 0.05, and 0.1, respectively. $\delta = 0.1$ yields the worst runtime since it does not allow NAMOA*+BO-DH to activate some landmarks that could be useful. Different line styles (solid, dashed, and dotted) represent different i-values of 50, 100, and 200, respectively. In almost every case, an i-value of 100 yields the best runtime performance since it balances the benefit and the computational overhead of attempting to activate new landmarks.

Evaluating BO-DHs on different graphs: We compared NAMOA* and NAMOA*+BO-DH on different graphs. For each graph, we evaluated all combinations of parameters $|L| \in \{64, 128, 256\}$, $\epsilon \in \{0.005, 0.01\}$, $i \in \{50, 100, 200, 500\}$, and $\delta \in \{0.01, 0.05, 0.1\}$ and report the best result for each graph.

Algorithm	Memory	t_{query}	$t_{eval\text{-}h}$	#exp
empty-32-32				
NAMOA*		1.37ms		899
NAMOA*+BO-DH	17.4MB	1.05ms	0.25ms	370
den520d				
NAMOA*		1.13s		430K
NAMOA*+BO-DH	7.2GB	0.45s	0.11s	55K
NY				
NAMOA*		0.46s		193K
NAMOA*+BO-DH	9.1GB	0.17s	0.01s	80K
BAY				
NAMOA*		0.79s		359K
NAMOA*+BO-DH	2.6GB	0.40s	0.16s	140K

Table 2: Experimental results for BO-DHs on different graphs. For each algorithm, we report the memory usage for the lookup table, the average runtime for solving an instance (in either milliseconds (ms) or seconds (s)), and the average number of expanded nodes.



(a) #exp (b) t_{query}

Figure 5: Experimental results on individual instances.

Table 2 summarizes the results. NAMOA*+BO-DH has smaller average runtime and node expansion than NAMOA* on all graphs. Fig. 5 shows the runtime (in (a)) and node expansion (in (b)) of NAMOA* and NAMOA*+BO-DH for each instance. BO-DH yields up to more than 10× reduction in node expansions and up to more than 8× reduction in runtime. For many instances on the road networks, BO-DHs do not yield any improvement in node expansion because no landmark is activated. However, since the grids have smaller sizes and are better covered by the landmarks, BO-DH yields smaller node expansions in almost all instances. In general, the improvements are more significant for more difficult instances (in the top right corners of the figures).

7 Conclusions

In this paper, we investigate using multi-valued heuristics to improve bi-objective search in explicit state space and propose BO-DHs, a bi-objective generalization of DHs. We propose several techniques to reduce the memory usage and computational overhead of BO-DH significantly. Future work includes combining BO-DHs with dimensionality reduction and generalizing BO-DHs to more than two objectives.

Acknowledgements

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, 2121028, and 2112533. The research was also supported by the United States-Israel Binational Science Foundation (BSF) under grant number 2021643 and Centro Nacional de Inteligencia Artificial CENIA, FB210017, BASAL, ANID. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or any government.

References

Ahmadi, S.; Tack, G.; Harabor, D. D.; and Kilby, P. 2021. Bi-objective Search with Bi-directional A*. In *Symposium on Combinatorial Search (SOCS)*, 142–144.

Bachmann, D.; Bökler, F.; Kopec, J.; Popp, K.; Schwarze, B.; and Weichert, F. 2018. Multi-Objective Optimisation Based Planning of Power-Line Grid Expansions. *ISPRS International Journal of Geo-Information*, 7(7): 258.

Bronfman, A.; Marianov, V.; Paredes-Belmar, G.; and Lüer-Villagra, A. 2015. The Maximin HAZMAT Routing Problem. *European Journal of Operational Research*, 241(1): 15–27.

Fu, M.; Kuntz, A.; Salzman, O.; and Alterovitz, R. 2019. Toward Asymptotically-Optimal Inspection Planning via Efficient Near-Optimal Graph Search. In *Robotics: Science and Systems (RSS)*.

Fu, M.; Salzman, O.; and Alterovitz, R. 2021. Computationally-Efficient Roadmap-Based Inspection Planning via Incremental Lazy Search. In *IEEE International Conference on Robotics and Automation (ICRA)*, 7449–7456.

Geißer, F.; Haslum, P.; Thiébaux, S.; and Trevizan, F. 2022. Admissible Heuristics for Multi-Objective Planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 100–109.

Goldberg, A. V.; Kaplan, H.; and Werneck, R. F. 2006. Reach for A*: Shortest Path Algorithms with Preprocessing. In *The Shortest Path Problem, Proceedings of a DIMACS Workshop*, 93–139.

Goldberg, A. V.; and Werneck, R. F. 2005. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, 26–40.

Goldenberg, M.; Sturtevant, N.; Felner, A.; and Schaeffer, J. 2011. The Compressed Differential Heuristic. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 25, 24–29.

Hernández, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; Koenig, S.; and Salzman, O. 2023. Simple and Efficient Bi-objective Search Algorithms via Fast Dominance Checks. *Artificial Intelligence*, 314: 103807.

Mandow, L.; and De La Cruz, J. L. P. 2010. Multiobjective A* Search with Consistent Heuristics. *Journal of the ACM*, 57(5): 1–25.

Maristany de las Casas, P.; Kraus, L.; Sedeño-Noda, A.; and Borndörfer, R. 2021. Targeted Multiobjective Dijkstra Algorithm. https://arxiv.org/abs/2110.10978.

Pulido, F. J.; Mandow, L.; and De la Cruz, J. L. P. 2015. Dimensionality Reduction in Multiobjective Shortest Path Search. *Computers & Operations Research*, 64: 60–70.

Ren, Z.; Zhan, R.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022. Enhanced Multi-Objective A* Using Balanced Binary Search Trees. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 162–170.

Sedeño-Noda, A.; and Colebrook, M. 2019. A Biobjective Dijkstra Algorithm. *European Journal of Operational Research*, 276(1): 106–118.

Sturtevant, N. R.; Felner, A.; Barrer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-Based Heuristics for Explicit State Spaces. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 609–614.

Zhang, H.; Salzman, O.; Kumar, T. S.; Felner, A.; Hernández, C.; and Koenig, S. 2022. A* pex: Efficient Approximate Multi-Objective Search on Graphs. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 394–403.