Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

A Simple and Fast Bi-Objective Search Algorithm

Carlos Hernández Ulloa,[†] William Yeoh,[‡] Jorge A. Baier,[*, §] Han Zhang,[⋆] Luis Suazo,[†] Sven Koenig[⋆]
[†]Departamento de Ciencias de la Ingeniería, Universidad Andrés Bello, Chile
[‡]Department of Computer Science and Engineering, Washington University in St. Louis, USA
[*]Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Chile
[§]Millennium Institute for Foundational Research on Data, Chile
[⋆]Department of Computer Science, University of Southern California, USA

Abstract

Many interesting search problems can be formulated as bi-objective search problems, that is, search problems where two kinds of costs have to be minimized, for example, travel distance and time for transportation problems. Bi-objective search algorithms have to maintain the set of undominated paths from the start state to each state to compute the set of paths from the start state to the goal state that are not dominated by some other path from the start state to the goal state (called the Pareto-optimal solution set). Each time they find a new path to a state s, they perform a *dominance check* to determine whether this path dominates any of the previously found paths to s or whether any of the previously found paths to s dominates this path. Existing algorithms do not perform these checks efficiently. On the other hand, our Bi-Objective A* (BOA*) algorithm requires only constant time per check. In our experimental evaluation, we show that BOA* can run an order of magnitude (or more) faster than state-of-the-art bi-objective search algorithms, such as NAMOA*, NAMOA*dr, Bi-Objective Dijkstra, and Bidirectional Bi-Objective Dijkstra.

Introduction

The A* algorithm (Hart, Nilsson, and Raphael 1968) is at the core of many heuristic search algorithms developed to solve shortest path problems due to its strong theoretical properties, especially when used in conjunction with consistent heuristic functions. In such problems, one has to find a path from a given start state to a given goal state that minimizes the path cost. However, there are often multiple kinds of path costs in real life. For example, government agencies that transport hazardous material need to find routes that do not only minimize the travel distance but also the risk of exposure for residents (Bronfman et al. 2015). Motivated by such applications, researchers have extended A* to solve multi-objective shortest path problems where one wants to find the set of Pareto-optimal paths from the start state to the goal state, that is, the optimal paths on the Pareto frontier. Two such state-of-the-art A* extensions are the *Multi-Objective A** (MOA*) (Stewart and White III 1991) and *New Approach for MOA** (NAMOA*) (Mandow and Pérez-de-la-Cruz 2010) algorithms.

These best-first multi-objective search algorithms differ from A* in various ways. The most relevant difference in the context of this paper is that the concept of optimality is now related to dominance since the set of Pareto-optimal paths is the set of paths that are not dominated by any path, where path p dominates path p' iff each kind of path cost of p is no larger than the corresponding kind of path cost of p' and at least one kind of path cost of p is smaller than the corresponding kind of path cost of p'. Since dominance checks are repeatedly performed throughout the execution of these algorithms, the time complexity of the checks plays a crucial role for their efficiency. For example, upon generating any node, they need to check if the newly found path to some state s is dominated by a previously found path to s and, if so, discard the newly found path. They also need to check whether a previously found path to s is dominated by the newly found path to s and, if so, discard the previously found path.

NAMOA* is inefficient at performing these checks. Pulido, Mandow, and Pérez-de-la-Cruz (2015) proposed an improvement, called NAMOA*dr. NAMOA*dr significantly improves the time complexity of some of the checks to *constant* time, but the time complexity of other checks remains *linear* in the size of the $Open$ list and the number of paths found to a given state.

In this paper, we address these limitations. Our *Bi-Objective A** (BOA*) algorithm prunes dominated paths more efficiently by exploiting that there are only two kinds of path costs and that the heuristic function is consistent. It performs *all* dominance checks in *constant time*, which we achieve by making some of the eager checks more efficient and converting the remaining eager check into a number of lazy checks, each of which can be performed in constant time. This improvement results in a significant speedup, especially for large instances.

Our extensive experimental results on road maps show that BOA* can run an order of magnitude (or more) faster than NAMOA*, NAMOA*dr, Bi-Objective Dijkstra, and Bidirectional Bi-Objective Dijkstra, especially for large instances. We conclude the paper by discussing how one might be able to improve and extend BOA*, including how to speed it up, find representative solutions on the Pareto frontier, find bounded-suboptimal solutions, and generalize it to problems with more than two kinds of path costs.

Notation and Terminology

A bi-objective *search graph* is a tuple $(S, E, \mathbf{c})$, where S is the finite set of *states*, $E \subseteq S \times S$ is the finite set of edges, and $\mathbf{c} : E \rightarrow \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$ is a *cost function* that associates a pair of non-negative real costs with each edge. $Succ(s) = \{t \in S \mid (s,t) \in E\}$ denotes the successors of state s.

A bi-objective *search problem instance* is a tuple $P = (S, E, \mathbf{c}, s_{start}, s_{goal})$, where $(S, E, \mathbf{c})$ is a search graph, $s_{start} \in S$ is the start state, and $s_{goal} \in S$ is the goal state.[1] A *path* from s_1 to s_n is a sequence of states $s_1, s_2, \ldots, s_n$ such that $(s_i, s_{i+1}) \in E$ for all $i \in \{1, \ldots, n-1\}$. Unless mentioned otherwise, $s_1 = s_{start}$. A path is a *solution* for instance P iff it is a path (from s_{start}) to s_{goal}.

Boldface font indicates pairs. p_1 denotes the first component of pair $\mathbf{p}$, and p_2 denotes its second component; that is, $\mathbf{p} = (p_1, p_2)$. The addition of two pairs $\mathbf{p}$ and $\mathbf{q}$ and the multiplication of a real-valued scalar k and a pair $\mathbf{p}$ are defined in the natural way, namely as $\mathbf{p}+\mathbf{q} = (p_1+q_1, p_2+q_2)$ and $k\mathbf{p} = (kp_1, kp_2)$, respectively. $\mathbf{p} \prec \mathbf{q}$ denotes that ($p_1 < q_1$ and $p_2 \leq q_2$) or ($p_1 = q_1$ and $p_2 < q_2$). In this case, we say that $\mathbf{p}$ *dominates* $\mathbf{q}$. $\mathbf{p} \leq \mathbf{q}$ denotes that $p_1 \leq q_1$ and $p_2 \leq q_2$. In this case, we say that $\mathbf{p}$ *weakly dominates* $\mathbf{q}$. $P \prec \mathbf{q}$ (resp. $P \leq \mathbf{q}$) for a set P of pairs denotes that there exists a $\mathbf{p} \in P$ such that $\mathbf{p} \prec \mathbf{q}$ (resp. $\mathbf{p} \leq \mathbf{q}$).

$\mathbf{c}(\pi) = \sum_{i=1}^{n-1} \mathbf{c}(s_i, s_{i+1})$ is the cost of path $\pi = s_1, \ldots, s_n$. $\pi \prec \pi'$ (resp. $\pi \leq \pi'$) for two paths π and π' denotes that $\mathbf{c}(\pi) \prec \mathbf{c}(\pi')$ (resp. $\mathbf{c}(\pi) \leq \mathbf{c}(\pi')$). In this case, we say that π *dominates* (resp. *weakly dominates*) π'.

Given an instance P, a *Pareto-optimal solution* π for P is a solution for P such that $\pi' \not\prec \pi$ for all solutions π' for P, that is, a Pareto-optimal solution is one that is not dominated by any solution. The Pareto-optimal solution set is the set of all Pareto-optimal solutions. We are interested in finding any maximal subset of the Pareto-optimal solution set such that any two solutions in the subset do not have the same cost and refer to this subset as the *cost-unique Pareto-optimal solution* set.

A *heuristic function* $\mathbf{h} : S \rightarrow \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$ is such that the h-value $\mathbf{h}(s)$ estimates the cost of a path from state s to the goal state. $\mathbf{h}$ is admissible iff $\mathbf{h}(s) \leq \mathbf{c}(\pi)$ for all states s and all paths π from s to the goal state, that is, both components of $\mathbf{h}$ are admissible for the corresponding components of the cost function. Similarly, $\mathbf{h}$ is consistent iff (1) $\mathbf{h}(s_{goal}) = (0, 0)$ and (2) $\mathbf{h}(s) \leq \mathbf{c}(s,t) + \mathbf{h}(t)$ for all $(s,t) \in E$. We assume that the reader is familiar with the properties of A* when used with a consistent heuristic function, for example, that the sequence of expanded nodes has monotonically non-decreasing f-values.

[1]We use a single goal state for simplicity since any search problem instance with multiple goal states can be transformed into one with a single goal state.

Best-First Bi-Objective Search

In this section, we describe how a Pareto-optimal solution set can be computed using best-first search.

Open List: We can compute the Pareto-optimal solution set with a modified version of A* that maintains an $Open$ list, containing the frontier of the search tree (that is, the generated but not yet expanded nodes), and, optionally, a $Closed$ list, containing the interior of the search tree (that is, the expanded nodes). A node is associated with a state, a g-value, an h-value, and an f-value and corresponds to a path to the state of a cost that is equal to the g-value. Different from A*, the g-, h-, and f-values are tuples rather than scalars. Also different from A*, the $Open$ list might contain different nodes with the same state, corresponding to different paths to the same state, since we need to compute the Pareto-optimal solution set rather than a single solution.

Node Selection: The algorithm repeatedly extracts a node from the $Open$ list. To guarantee optimality, the f-value of the extracted node must not be dominated by the f-value of any node in the $Open$ list.

Solution Recording: When the algorithm extracts a node with the goal state, the path corresponding to the node is a solution. Different from A*, the algorithm cannot terminate and return this solution since it has to compute the Pareto-optimal solution set. Thus, it checks whether this solution is dominated by a previously found solution. If not, then it adds this solution to the solution set and removes all solutions from the solution set that are dominated by this solution. In both cases, it continues the search.

Node Expansion: When the algorithm extracts a node with a non-goal state, it expands the extracted node. Let the extracted node have state s. The algorithm then generates the child nodes of the extracted node, one for each successor t of s, by adding them to the $Open$ list. It terminates when the $Open$ list is empty and returns the solution set.

Efficiency: We can improve the efficiency of the algorithm by performing the dominance checks not once it has found a solution but earlier. In particular, we do not need to generate a child node with state t of an extracted node if the f-value of the child node (which is a lower bound on the costs of all solutions that complete the path that the child node corresponds to) is dominated by the f-value (that is, cost) of a solution in the solution set or by the f-value of a node with state t that has already been generated (corresponding to a path to t that has already been found). In addition, we can remove all paths to t from the $Open$ list whose f-values are dominated by the f-value of the newly found path to t. If t is the goal state, we also have to remove all solutions from the solution set whose f-values (that is, costs) are dominated by the f-value (that is, cost) of the newly found solution.

The NAMOA* Algorithm

NAMOA* (Mandow and Pérez-de-la-Cruz 2010) is a best-first multi-objective search algorithm that provides the foundation for most multi-objective search algorithms. Algorithm 1 shows its pseudocode for bi-objective search problems. It takes as input a bi-objective search problem and a consistent heuristic function and computes the Pareto-optimal solution set. We describe its key elements in the following.

Variables: Each node in the $Open$ list is a triple of the form $(s, \mathbf{g}_s, \mathbf{f}_s)$ with state s, g-value $\mathbf{g}_s$, and f-value $\mathbf{f}_s$ and corresponds to a path to s of cost $\mathbf{g}_s$. In addition, NAMOA* maintains parents. Different from A*, a parent is a set of g-values of some of the predecessors of s (rather than a single predecessor) and is associated with g-value $\mathbf{g}_s$ (rather than state s). Also different from A*, NAMOA* also maintains two sets of g-values for state s, namely $\mathbf{G}_{cl}(s)$, which contains the g-values of all expanded nodes with state s, and $\mathbf{G}_{op}(s)$, which contains the g-values of all generated but not yet expanded nodes with state s.

Node Selection: NAMOA* always extracts a node from the $Open$ list whose f-value is not dominated by the f-value of any node in the $Open$ list. Such a node can be identified efficiently for bi-objective search problems as a node in the $Open$ list with the lexicographically smallest f-value (f_1, f_2) of all nodes in the $Open$ list (Line 8). To see why this is correct, let (f'_1, f'_2) be the f-value of any node in the $Open$ list. Then, either (1) $f_1 = f'_1$ and $f_2 \leq f'_2$ or (2) $f_1 < f'_1$. In both cases, $(f'_1, f'_2) \not\prec (f_1, f_2)$; that is, (f_1, f_2) is not dominated by the f-value of any node in the $Open$ list. Consequently, the nodes in the $Open$ list should be ordered in increasing lexicographic order of their f-values.

Solution Recording: When NAMOA* extracts a node with the goal state, it has found an undominated solution. In this case, it adds the g-value of the node to the solution set and removes all nodes from the $Open$ list whose f-values are dominated by the f-value of the node (Lines 10-13).

Node Expansion: When NAMOA* extracts a node with a non-goal state, it expands the extracted node $(s, \mathbf{g}_s, \mathbf{f}_s)$ by calculating its child nodes $(t, \mathbf{g}_t, \mathbf{f}_t)$, one for each successor t of state s. If it has generated a node with state t and g-value $\mathbf{g}_t$ before, then it adds $\mathbf{g}_s$ to the parent set $parent(\mathbf{g}_t)$ (Lines 16-18) (which corresponds to recording another path to t of cost $\mathbf{g}_t$ and is necessary since NAMOA* computes the Pareto-optimal solution set rather than a single solution). In this case, it does not add the child node to the $Open$ list. Neither does it add the child node to the $Open$ list if $\mathbf{g}_t$ is dominated by the g-value of a generated node with state t (Lines 19-20) (which corresponds to pruning the newly found path to t since it is dominated by another path to t that has already been found). Neither does it add the child node to the $Open$ list if the f-value $\mathbf{f}_t$ is dominated by the f-value (that is, g-value and cost) of a solution in the solution set (Lines 22-23) (which corresponds to pruning the newly found path to t since it is dominated by a solution that has already been found). Otherwise, it generates the child node by adding it to the $Open$ list, adding $\mathbf{g}_t$ to $\mathbf{G}_{op}(t)$, making $\mathbf{g}_s$ the only g-value in the parent set $parent(\mathbf{g}_t)$ (which corresponds to recording the first path to t of cost $\mathbf{g}_t$), and removing all references to paths to t from the $Open$ list, $\mathbf{G}_{op}(t)$, and $\mathbf{G}_{cl}(t)$

Algorithm 1: NAMOA*

Input : A search problem $(S, E, \mathbf{c}, s_{start}, s_{goal})$ and a consistent heuristic function $\mathbf{h}$
Output: The Pareto-optimal solution set

1 $sols \leftarrow \emptyset$
2 **for each** $s \in S$ **do**
3 $\quad \mathbf{G}_{op}(s) \leftarrow \emptyset$; $\mathbf{G}_{cl}(s) \leftarrow \emptyset$
4 $\mathbf{G}_{op}(s) \leftarrow \{(0,0)\}$
5 $parent((0,0)) \leftarrow \emptyset$
6 Initialize $Open$ and add $(s_{start}, (0,0), \mathbf{h}(s_{start}))$ to it
7 **while** $Open \neq \emptyset$ **do**
8 $\quad$ Remove a node $(s, \mathbf{g}_s, \mathbf{f}_s)$ from $Open$ with the lexicographically smallest f-value of all nodes in $Open$
9 $\quad$ Remove $\mathbf{g}_s$ from $\mathbf{G}_{op}(s)$ and add it to $\mathbf{G}_{cl}(s)$
10 $\quad$ **if** $s = s_{goal}$ **then**
11 $\quad\quad$ Add $\mathbf{g}_s$ to $sols$
12 $\quad\quad$ Remove all nodes $(u, \mathbf{g}_u, \mathbf{f}_u)$ with $\mathbf{f}_s \prec \mathbf{f}_u$ from $Open$
13 $\quad\quad$ **continue**
14 $\quad$ **for each** $t \in \texttt{Succ}(s)$ **do**
15 $\quad\quad$ $\mathbf{g}_t \leftarrow \mathbf{g}_s + \mathbf{c}(s, t)$
16 $\quad\quad$ **if** $\mathbf{g}_t \in \mathbf{G}_{op}(t) \cup \mathbf{G}_{cl}(t)$ **then**
17 $\quad\quad\quad$ Add $\mathbf{g}_s$ to $parent(\mathbf{g}_t)$
18 $\quad\quad\quad$ **continue**
19 $\quad\quad$ **if** $\mathbf{G}_{op}(t) \cup \mathbf{G}_{cl}(t) \prec \mathbf{g}_t$ **then**
20 $\quad\quad\quad$ **continue**
21 $\quad\quad$ $\mathbf{f}_t \leftarrow \mathbf{g}_t + \mathbf{h}(t)$
22 $\quad\quad$ **if** $sols \prec \mathbf{f}_t$ **then**
23 $\quad\quad\quad$ **continue**
24 $\quad\quad$ Remove all g-values $\mathbf{g}'_t$ from $\mathbf{G}_{op}(t)$ that are dominated by $\mathbf{g}_t$ and remove their corresponding nodes $(t, \mathbf{g}'_t, \mathbf{f}'_t)$ from $Open$
25 $\quad\quad$ Remove all g-values from $\mathbf{G}_{cl}(t)$ that are dominated by $\mathbf{g}_t$
26 $\quad\quad$ $parent(\mathbf{g}_t) \leftarrow \{\mathbf{g}_s\}$
27 $\quad\quad$ Add $\mathbf{g}_t$ to $\mathbf{G}_{op}(t)$
28 $\quad\quad$ Add $(t, \mathbf{g}_t, \mathbf{f}_t)$ to $Open$
29 **return** $sols$

that are dominated by the newly found path to t (Lines 24-28). It terminates when the $Open$ list is empty and returns the solution set (Line 29).

The NAMOA*dr Algorithm

Some of the operations of NAMOA* are time-consuming since they perform dominance checks that involve either the f-values (Lines 12 and 22) or g-values (Lines 24-25) and require it to iterate over a number of elements proportional to $|\mathbf{G}_{op}(t)|$, $|\mathbf{G}_{cl}(t)|$, $|Open|$, or $|sols|$. Pulido, Mandow, and Pérez-de-la-Cruz (2015) (in short: PMP) improved NAMOA* to NAMOA*dr by proving that, if NAMOA* (A1) uses a consistent heuristic function and (A2) always extracts a node with the lexicographically smallest f-value of all nodes in the $Open$ list, then the following theorem holds for bi-objective search problems:

Theorem 1 (Pulido, Mandow, and Pérez-de-la-Cruz 2015) Assume that A1 and A2 hold and let $(s, \mathbf{g}, \mathbf{f})$ be a newly extracted node. Then, $\mathbf{G}_{cl}(t) \prec \mathbf{g_t}$ (Line 19) and $sols \prec \mathbf{f_t}$ (Line 22) can be decided in constant time for bi-objective search problems.

More specifically, checking whether $\mathbf{G}_{cl}(t) \prec \mathbf{g_t}$ can be done as follows: $\mathbf{G}_{cl}(t) \prec (g_1, g_2)$ iff $g^{\min} < g_2$, where $g^{\min}$ is the minimum of the g_2-values in $\mathbf{G}_{cl}(t)$. Checking whether $sols \prec \mathbf{f_t}$ can be done analogously. NAMOA*dr uses these insights to implement Lines 19 and 22 in constant time, except that Line 19 still needs to iterate over a number of g-values proportional to $|\mathbf{G}_{op}(t)|$ to check whether $\mathbf{G}_{op}(t) \prec \mathbf{g_t}$.

Our Bi-Objective A* (BOA*) Algorithm

The improvements to NAMOA* proposed by PMP remove some, but not all, of its most time-consuming operations since it still iterates over a number of nodes proportional to $|Open|$ on Line 12, a number of g-values proportional to $|\mathbf{G}_{op}(t)|$ on Lines 19 and 24, and a number of g-values proportional to $|\mathbf{G}_{cl}(t)|$ on Line 25. In this section, we therefore describe our *Bi-Objective A** (BOA*) algorithm, a best-first bi-objective search algorithm. Our primary design objective is to perform all dominance checks in constant time. We use Theorem 1 and additional insights (1) to avoid having to maintain the sets $\mathbf{G}_{op}(s)$ and $\mathbf{G}_{cl}(s)$ for all states s and thus not having to perform any of the eager checks on Lines 24-25 to remove g-values from these sets and (2) to make the eager check on Line 19 more efficient by maintaining a value $g_2^{\min}(s)$ for each state s, which is the smallest g_2-value of any expanded node with state s. The remaining eager checks on Lines 12 and 24 remove nodes from the $Open$ list. We convert these checks into a number of lazy checks, each of which can be performed in constant time, by not removing these nodes from the $Open$ list (which is time-consuming but might result in fewer heap percolations) but by performing the checks when nodes get extracted from the $Open$ list and then not expanding these nodes. A secondary design objective is to make the presentation of BOA* similar to that of modern descriptions of A*, such as those in (Edelkamp and Schrödl 2011), thereby making it potentially easier to understand and implement. Another secondary design objective is to compute the cost-unique Pareto-optimal set rather than the Pareto-optimal set since it is sufficient for our purposes to compute one representative solution for all cost-identical and thus equally good solutions.

The $Open$ list of BOA* contains *nodes*, which are akin to the *labels* commonly used in the operations research literature (Raith and Ehrgott 2009). Each node x has a state $s(x)$, a g-value $\mathbf{g}(x)$, an f-value $\mathbf{f}(x)$, and a parent $parent(x)$ and corresponds to a path to $s(x)$ of cost $\mathbf{g}(x)$. The parent is a single node.

Algorithm 2 shows the pseudocode of BOA*. It takes as input a bi-objective search problem and a consistent heuristic function and computes the cost-unique Pareto-optimal solution set. In each iteration, it extracts a node x from the

Algorithm 2: Bi-Objective A* (BOA*)

Input : A search problem $(S, E, \mathbf{c}, s_{start}, s_{goal})$ and a consistent heuristic function $\mathbf{h}$
Output: A cost-unique Pareto-optimal solution set
1 $sols \leftarrow \emptyset$
2 **for each** $s \in S$ **do**
3 $\quad g_2^{\min}(s) \leftarrow \infty$
4 $x \leftarrow$ new node with $s(x) = s_{start}$
5 $\mathbf{g}(x) \leftarrow (0, 0)$
6 $parent(x) \leftarrow$ null
7 $\mathbf{f}(x) \leftarrow (h_1(s_{start}), h_2(s_{start}))$
8 Initialize $Open$ and add x to it
9 **while** $Open \neq \emptyset$ **do**
10 $\quad$ Remove a node x from $Open$ with the lexicographically smallest f-value of all nodes in $Open$
11 $\quad$ **if** $g_2(x) \geq g_2^{\min}(s(x)) \vee f_2(x) \geq g_2^{\min}(s_{goal})$ **then**
12 $\quad\quad$ **continue**
13 $\quad g_2^{\min}(s(x)) \leftarrow g_2(x)$
14 $\quad$ **if** $s(x) = s_{goal}$ **then**
15 $\quad\quad$ Add x to $sols$
16 $\quad\quad$ **continue**
17 $\quad$ **for each** $t \in \texttt{Succ}(s(x))$ **do**
18 $\quad\quad y \leftarrow$ new node with $s(y) = t$
19 $\quad\quad \mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{c}(s(x), t)$
20 $\quad\quad parent(y) \leftarrow x$
21 $\quad\quad \mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$
22 $\quad\quad$ **if** $g_2(y) \geq g_2^{\min}(t) \vee f_2(y) \geq g_2^{\min}(s_{goal})$ **then**
23 $\quad\quad\quad$ **continue**
24 $\quad\quad$ Add y to $Open$
25 **return** $sols$

$Open$ list with the lexicographically smallest f-value of all nodes in the $Open$ list (Line 10). It does not expand the node if its g_2-value is at least $g_2^{\min}(s(x))$ or its f_2-value is at least $g_2^{\min}(s_{goal})$ (Lines 11-12). Otherwise, it updates $g_2^{\min}(s(x))$ (Line 13) and expands the node. If $s(x)$ is the goal state, then BOA* has found an undominated solution and adds node x to the solution set $sols$ (Lines 14-16). Otherwise, it calculates the child nodes of node x (Lines 18-21). It does not add a child node y to the $Open$ list if its g_2-value is at least $g_2^{\min}(s(y))$ or its f_2-value is at least $g_2^{\min}(s_{goal})$ (Lines 22-23). Otherwise, it generates the child node by adding it to the $Open$ list (Line 24). It terminates when the $Open$ list is empty and returns the solution set (Line 25).

Figure 1 shows a small example of the operation of BOA*. We use the perfect distances as h-values, which can be computed with Dijkstra's algorithm. Table 1 shows a trace of the $Open$ list and changes to $g_2^{\min}$ in each iteration of BOA*. In the table and the text below, the triple $(s, \mathbf{g}, \mathbf{f})$ refers to a node with state s, g-value $\mathbf{g}$, and f-value $\mathbf{f}$.

- In Iteration 1, the node $(s_{start}, (0, 0), (3, 6))$ is expanded, and its three child nodes with states s_1, s_2, and s_3 are added to the $Open$ list.
- In Iteration 2, node $(s_2, (1, 5), (3, 9))$ is expanded, and its

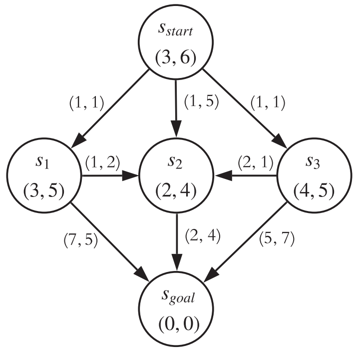

Figure 1: Example search graph. The pair of numbers inside each state is its h-value.

Iteration	*Open* list $((s(x), g(x), f(x)))$	Update of $g_2^{\min}(s(x))$
1	$(s_{start}, (0,0), (3,6)) \leftarrow$	$g_2^{\min}(s_{start}) = 0$
2	$(s_1, (1,1), (4,6))$ $(s_2, (1,5), (3,9)) \leftarrow$ $(s_3, (1,1), (5,6))$	$g_2^{\min}(s_2) = 5$
3	$(s_1, (1,1), (4,6))$ $(s_3, (1,1), (5,6))$ $(s_{goal}, (3,9), (3,9)) \leftarrow$	$g_2^{\min}(s_{goal}) = 9$
4	$(s_1, (1,1), (4,6)) \leftarrow$ $(s_3, (1,1), (5,6))$	$g_2^{\min}(s_1) = 1$
5	$(s_3 (1,1), (5,6))$ $(s_{goal}, (8,6), (8,6))$ $(s_2, (2,3), (4,7)) \leftarrow$	$g_2^{\min}(s_2) = 3$
6	$(s_3, (1,1), (5,6))$ $(s_{goal}, (4,7), (4,7)) \leftarrow$ $(s_{goal}, (8,6), (8,6))$	$g_2^{\min}(s_{goal}) = 7$
7	$(s_3, (1,1), (5,6)) \leftarrow$ $(s_{goal}, (8,6), (8,6))$	$g_2^{\min}(s_3) = 1$
8	$(s_2, (3,2), (5,6)) \leftarrow$ $(s_{goal}, (8,6), (8,6))$	$g_2^{\min}(s_2) = 2$
9	$(s_{goal}, (5,6), (5,6)) \leftarrow$ $(s_{goal}, (8,6), (8,6))$	$g_2^{\min}(s_{goal}) = 6$
10	$(s_{goal}, (8,6), (8,6)) \leftarrow$	
11	empty	

Table 1: Trace of the *Open* list and $g_2^{\min}(s(x))$ in each iteration of BOA*. $\leftarrow$ marks the node that is extracted in that iteration.

child node with state s_{goal} is added to the *Open* list.

- In Iteration 3, node $(s_{goal}, (3,9), (3,9))$ is expanded, and the first undominated solution is found.
- In Iteration 4, node $(s_1, (1,1), (4,6))$ is expanded, and its two child nodes with states s_2 and s_{goal} are added to the *Open* list.
- In Iteration 5, node $(s_2, (2,3), (4,7))$ is expanded, and its child node with state s_{goal} is added to the *Open* list.
- In Iteration 6, node $(s_{goal}, (4,7), (4,7))$ is expanded, and the second undominated solution is found.
- In Iteration 7, node $(s_3, (1,1), (5,6))$ is expanded, and its child node with state s_2 is added to the *Open* list. Its child node $(s_{goal}, (6,8), (6,8))$ is not added to the *Open* list because $f_2(s_{goal}) = 8 \geq 7 = g_2^{\min}(s_{goal})$.
- In Iteration 8, node $(s_2, (3,2), (5,6))$ is expanded, and its child node with state s_{goal} is added to the *Open* list.
- In Iteration 9, node $(s_{goal}, (5,6), (5,6))$ is expanded, and the third undominated solution is found.
- In Iteration 10, node $(s_{goal}, (8,6), (8,6))$ is extracted but not expanded because $f_2(s_{goal}) = 6 \geq 6 = g_2^{\min}(s_{goal})$.
- Finally, in Iteration 11, the *Open* list is empty, and BOA* returns the three undominated solutions found.

Theoretical Results for BOA*

We assume that heuristic function $\mathbf{h}$ is consistent. We say that a node x_1 dominates (resp. weakly dominates) a node x_2 iff the g-value of node x_1 dominates (resp. weakly dominates) the g-value of node x_2.

Lemma 1 *Each generated (or about to be generated but pruned) node x has f_1- and f_2-values that are no smaller than the f_1- and f_2-values, respectively, of its parent node p.*

Proof Sketch: Since the h-values are consistent, $c_1(s(p), s(x)) + h_1(s(x)) \geq h_1(s(p))$. Therefore, we get:

$$\begin{aligned} f_1(x) &= g_1(x) + h_1(s(x)) \\ &= g_1(p) + c_1(s(p), s(x)) + h_1(s(x)) \\ &\geq g_1(p) + h_1(s(p)) \\ &= f_1(p) \end{aligned}$$

The same proof strategy yields $f_2(x) \geq f_2(p)$. □

Lemma 2 *The sequences of extracted nodes and of expanded nodes have monotonically non-decreasing f_1-values.*

Proof Sketch: BOA* extracts the node from the *Open* list with the lexicographically smallest f-value of all nodes in the *Open* list (Line 10). This node has the smallest f_1-value of all nodes in the *Open* list. Since generated nodes that are added to the *Open* list have f_1-values that are no smaller than those of their expanded parent nodes (Lemma 1), the sequence of extracted nodes has monotonically non-decreasing f_1-values. Since nodes are expanded in the same order in which they are extracted, the sequence of expanded nodes also has monotonically non-decreasing f_1-values. □

Lemma 3 *The sequence of expanded nodes with the same state has strictly monotonically decreasing f_2-values.*

Proof Sketch: Assume for a proof by contradiction that BOA* expands node x_1 with state s before node x_2 with state s, that it expands no node with state s after node x_1 and before node x_2, and that $f_2(x_1) \leq f_2(x_2)$. Then, $g_2(x_1) + h_2(s) = f_2(x_1) \leq f_2(x_2) = g_2(x_2) + h_2(s)$. Thus, $g_2(x_1) \leq g_2(x_2)$. After node x_1 is expanded and before node x_2 is expanded, $g_2^{\min}(s) = g_2(x_1)$ (Line 13). Combining both (in)equalities yields $g_2^{\min}(s) \leq g_2(x_2)$, which is the first pruning condition on Line 11. Therefore, node x_2 is not expanded, which contradicts the assumption. □

Lemma 4 *The sequence of expanded nodes with the same state has strictly monotonically increasing f_1-values.*

Proof Sketch: Since the sequence of expanded nodes has monotonically non-decreasing f_1-values (Lemma 2), the sequence of expanded nodes with the same state also has monotonically non-decreasing f_1-values. Assume for a proof by contradiction that BOA* expands node x_1 with state s before node x_2 with state s, that it expands no node with state s after node x_1 and before node x_2, and that $f_1(x_1) = f_1(x_2)$. We distinguish two cases:

- Node x_2 is in the *Open* list when BOA* expands node x_1: When BOA* expands node x_1, node x_1 has the lexicographically smallest f-value of all nodes in the *Open* list. Since $f_1(x_1) = f_1(x_2)$, it follows that $f_2(x_1) \le f_2(x_2)$, which contradicts Lemma 3.
- Node x_2 is not in the *Open* list when BOA* expands node x_1: BOA* thus generates node x_2 after it expands node x_1. Thus, there is a node x_3 in the *Open* list when BOA* expands node x_1 that is expanded after node x_1 (or is equal to it) and before node x_2 and becomes an ancestor node of node x_2 in the search tree. Since the sequence of expanded nodes has monotonically non-decreasing f_1-values (Lemma 2) and $f_1(x_1) = f_1(x_2)$, $f_1(x_1) = f_1(x_3) = f_1(x_2)$. When BOA* expands node x_1, node x_1 has the lexicographically smallest f-value of all nodes in the *Open* list. Since $f_1(x_1) = f_1(x_3)$, it follows that $f_2(x_1) \le f_2(x_3)$. Since each node has an f_2-value that is no smaller than the f_2-values of its ancestor nodes (Lemma 1), $f_2(x_3) \le f_2(x_2)$. Combining both inequalities yields $f_2(x_1) \le f_2(x_2)$, which contradicts Lemma 3. □

Lemma 5 *Expanded nodes with the same state do not weakly dominate each other.*

Proof Sketch: Assume that BOA* expands node x_1 with state s before node x_2 with state s. Since the sequence of expanded nodes with the same state has strictly monotonically decreasing f_2-values (Lemma 3), $f_2(x_1) > f_2(x_2)$. It follows that $g_2(x_1)+h(s) = f_2(x_1) > f_2(x_2) = g_2(x_2)+h(s)$ and thus $g_2(x_1) > g_2(x_2)$. Since the sequence has strictly monotonically increasing f_1-values (Lemma 4), the same reasoning yields $g_1(x_1) < g_1(x_2)$. According to the two inequalities, nodes x_1 and x_2 do not weakly dominate each other. □

Lemma 6 *If node x_1 with state s is weakly dominated by node x_2 with state s, then each node with the goal state in the subtree of the search tree rooted at node x_1 is weakly dominated by a node with the goal state in the subtree rooted at node x_2.*

Proof Sketch: Since node x_1 is weakly dominated by node x_2, $g_1(x_2) \le g_1(x_1)$. Assume that node x_3 is a node with the goal state in the subtree of the search tree rooted at node x_1. Let the sequence of states of the nodes along a branch of the search tree from the root node to node x_1 be $s_1, \dots, s_i$ (with $s_1 = s_{start}$ and $s_i = s$), the sequence of states of the nodes along a branch of the search tree from the root node to node x_2 be $s'_1, \dots, s'_j$ (with $s'_1 = s_{start}$ and $s'_j = s$), and the sequence of states of the nodes along a branch of the search tree from node x_1 to node x_3 be $\pi = s_i, \dots, s_k$ (with $s_k = s_{goal}$). Then, there is a node x_4 with the goal state in the subtree rooted at node x_2 such that the sequence of states of the nodes along a branch of the search tree from the root node to node x_4 is $s'_1, \dots, s'_j, s_{i+1}, \dots, s_k$. Since $g_1(x_2) \le g_1(x_1)$, it follows that $g_1(x_4) = g_1(x_2)+c_1(\pi) \le g_1(x_1) + c_1(\pi) = g_1(x_3)$ and thus $g_1(x_4) \le g_1(x_3)$. The same proof strategy yields $g_2(x_4) \le g_2(x_3)$. Combining both inequalities yields that node x_3 is weakly dominated by node x_4. □

Lemma 7 *When BOA* prunes a node x_1 with state s (on Line 11 or 22) and this prevents it in the future from adding a node x_2 (with the goal state) to the solution set (on Line 15), then it can still add in the future a node (with the goal state) that weakly dominates node x_2 (on Line 15).*

Proof Sketch: We prove the statement by induction on the number of pruned nodes so far, including node x_1. If the number of pruned nodes is zero, then the lemma trivially holds. Now assume that the number of pruned nodes is $n+1$ and the lemma holds for $n \ge 0$. We distinguish three cases:

- BOA* prunes node x_1 on Line 11 because of the (first) pruning condition $g_2(x_1) \ge g_2^{\min}(s)$. Then, BOA* has expanded a node x_4 with state s previously such that $g_2^{\min}(s) = g_2(x_4)$ since otherwise $g_2^{\min}(s) = \infty$ and the pruning condition could not hold. Combining both (in)equalities yields $g_2(x_1) \ge g_2(x_4)$. Since $f_1(x_1) \ge f_1(x_4)$ (Lemma 2), $g_1(x_1) + h(s) = f_1(x_1) \ge f_1(x_4) = g_1(x_4)+h(s)$ and thus $g_1(x_1) \ge g_1(x_4)$. Combining both inequalities yields that node x_1 is weakly dominated by node x_4 and thus each node with the goal state in the subtree rooted at node x_1, including node x_2, is weakly dominated by a node x_5 with the goal state in the subtree rooted at node x_4 (Lemma 6). In case BOA* has pruned a node that prevents it in the future from adding node x_5 to the solution set, then it can still add in the future a node (with the goal state) that weakly dominates node x_5 and thus also node x_2 (induction assumption).
- BOA* prunes node x_1 on Line 11 because of the (second) pruning condition $f_2(x_1) \ge g_2^{\min}(s_{goal})$. Then, BOA* has expanded a node x_4 with the goal state previously such that $g_2^{\min}(s_{goal}) = g_2(x_4)$ since otherwise $g_2^{\min}(s_{goal}) = \infty$ and the pruning condition could not hold. Combining both (in)equalities yields that $f_2(x_1) \ge g_2(x_4)$. Since node x_1 is an ancestor node of node x_2 in the search tree, $f_2(x_2) \ge f_2(x_1)$ (Lemma 1). Combining both inequalities yields $g_2(x_2) = f_2(x_2) \ge g_2(x_4)$. Since node x_1 is an ancestor node of node x_2 in the search tree, $g_1(x_2) = f_1(x_2) \ge f_1(x_1)$ (Lemma 1). Since $f_1(x_1) \ge f_1(x_4)$ (Lemma 2), it follows that $g_1(x_2) \ge f_1(x_1) \ge f_1(x_4) = g_1(x_4)$. Combining $g_1(x_2) \ge g_1(x_4)$ and $g_2(x_2) \ge g_2(x_4)$ yields that node x_2 is weakly dominated by node x_4 (with the goal state). In case BOA* has pruned a node that prevents it in the future from adding node x_4 to the solution set, then it can still add in the future a node (with the goal state) that weakly dominates

node x_4 and thus also node x_2 (induction assumption).

- BOA* prunes node x_1 on Line 22 because of the pruning condition $g_2(x_1) \geq g_2^{\min}(s)$ or $f_2(x_1) \geq g_2^{\min}(s_{goal})$. The proofs of Case (1) or Case (2), respectively, apply unchanged except that $f_1(x_1) \geq f_1(x_4)$ now holds for a different reason. Let node x_3 be the node that BOA* expands when it executes Line 22. Combining $f_1(x_1) \geq f_1(x_3)$ (Lemma 1) and $f_1(x_3) \geq f_1(x_4)$ (Lemma 2) yields $f_1(x_1) \geq f_1(x_4)$. □

Theorem 1 *BOA* computes a cost-unique Pareto-optimal solution set.*

Proof Sketch: Let the path of a node x (and the solution of a node x with the goal state) be the sequence of states of the nodes along a branch of the search tree from the root node to node x. Then, the g-value of node x is the cost of the path (or the solution). Since the costs are non-negative and expanded nodes with the same state do not weakly dominate each other (Lemma 5), the paths of the expanded nodes are cycle-free. Since there are only a finite number of cycle-free paths, there are only a finite number of expanded nodes and thus only a finite number of generated nodes that are put into the $Open$ list. Since one node is extracted from the $Open$ list during each iteration, there are only a finite number of iterations and BOA* terminates. Now consider any non-empty set X of all nodes whose solutions are Pareto-optimal solutions of a given but arbitrary cost **c**. When BOA* is prevented in the future from adding a node $x_1 \in X$ to the solution set, it can still add in the future a node x_2 (with the goal state) that weakly dominates node x_1 (Lemma 7). Thus, $x_2 \in X$, which implies that BOA* is never prevented from adding all nodes in X to the solution set. The computed solution set is thus a superset of a cost-unique Pareto-optimal solution set P. Since BOA* can add only expanded nodes to the solution set and expanded nodes with the goal state do not weakly dominate each other (Lemma 5), the computed solution set cannot contain solutions that are not Pareto-optimal or have the same cost as other solutions in the computed solution set. Thus, it is exactly the cost-unique Pareto-optimal solution set P. □

Experimental Results

Setup: We compare Bi-Objective A* (BOA*), NAMOA*dr (Pulido, Mandow, and Pérez-de-la-Cruz 2015), BOA* with standard linear-time dominance checking (sBOA*), Bi-Objective Dijkstra (BDijkstra), and Bidirectional Bi-Objective Dijkstra (BBDijkstra) (Sedeño-Noda and Colebrook 2019). We use the C implementations provided by the authors for BBDijkstra and BDijkstra (Sedeño-Noda and Colebrook 2019). We implement BOA*, sBOA*, and NAMOA*dr from scratch in C using a standard binary heap for the $Open$ list. We use the BOA* implementation for the other implementations as well. We run all experiments on a 2.20GHz Intel(R) Xeon(R) CPU Linux machine with 128GB of RAM. We use road maps from the 9th DIMACS Implementation Challenge: Shortest Path[2]. The cost components represent travel distances (c_1)

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

New York City (NY)				
264,346 states, 730,100 edges, $\|sols\|$ = 199 on average				
	Solved	Average	Max	Min
NAMOA*	50/50	157.17	1,936.36	**0.02**
sBOA*	50/50	9.75	148.65	0.10
NAMOA*dr	50/50	0.65	4.99	0.11
BOA*	50/50	**0.32**	**1.95**	0.11
BBDijkstra	50/50	1.94	23.43	0.26
BDijkstra	50/50	2.55	21.16	0.17

San Francisco Bay (BAY)				
321,270 states, 794,830 edges, $\|sols\|$ = 119 on average				
	Solved	Average	Max	Min
NAMOA*	50/50	58.87	1,474.76	**0.02**
sBOA*	50/50	3.38	120.57	0.12
NAMOA*dr	50/50	0.38	6.08	0.12
BOA*	50/50	**0.29**	**4.17**	0.12
BBDijkstra	50/50	0.87	9.61	0.28
BDijkstra	50/50	1.83	33.39	0.22

Colorado (COL)				
435,666 states, 1,042,400 edges, $\|sols\|$ = 427 on average				
	Solved	Average	Max	Min
NAMOA*	48/50	476.26	3,551.32	**0.08**
sBOA*	50/50	38.88	1,141.78	0.17
NAMOA*dr	50/50	2.16	57.40	0.17
BOA*	50/50	**0.79**	**15.26**	0.17
BBDijkstra	50/50	4.79	83.07	0.41
BDijkstra	50/50	7.78	135.24	0.29

Florida (FL)				
1,070,376 states, 2,712,798 edges, $\|sols\|$ = 739 on average				
	Solved	Average	Max	Min
NAMOA*	43/50	812.48	3,298.90	1.42
sBOA*	46/50	349.64	1,238.25	**0.43**
NAMOA*dr	50/50	19.66	329.79	**0.43**
BOA*	50/50	**4.59**	**60.54**	**0.43**
BBDijkstra	50/50	91.36	1,772.48	1.11
BDijkstra	50/50	158.33	2,722.69	0.77

Table 2: Runtime (in seconds) on 50 instances of the specified road map. When an algorithm times out after 3,600 seconds, we use 3,600 seconds in the calculation of the average.

and times (c_2). The h-values are the exact travel distances and times to the goal state, computed with Dijkstra's algorithm. It takes 75 milliseconds to compute the h-values for the largest road map. The reported runtimes include this computation. All algorithms obtain the same number of solutions for all instances used in the experiments, implying that no two Pareto-optimal solutions have the same cost.

Results: We compare the runtimes of the five algorithms on 50 instances each of 4 road maps from the USA used by Machuca and Mandow (2012). Table 2 shows the name of the road map, the number of states and edges of the map, and the average number of Pareto-optimal solutions. For each algorithm, it shows the number of instances solved within a runtime limit of 3,600 seconds as well as the average, maximum, and minimum runtimes (in seconds). We include the results for NAMOA* reported by Machuca and Mandow

#	$Start$	$Goal$	$\|sols\|$	(1)	(2)	(3)
1	1941792	785069	27	0.97	1.02	1.0
2	207871	3619	419	0.96	1.24	13.7
3	1137220	991262	1947	0.96	4.11	23.9
4	1836318	1792612	4072	0.96	8.39	43.7

Table 3: Four instances of LKS. (1) Ratio of generated nodes (NAMOA*dr/BOA*). (2) Ratio of runtimes (NAMOA*dr/BOA*). (3) Number of *op-pruning* operations per generated node for NAMOA*dr.

$\|sols\|$ = 3,876 on average				
	Solved	Average	Max	Min
BOA* (f_1, f_2)	91/100	478.72	2,505	1.30
BOA* (f_2, f_1)	93/100	**383.79**	**2,059**	**1.26**

Table 4: Runtime (in seconds) on 100 instances of LKS. When an algorithm times out after 3,600 seconds, we use 3,600 seconds in the calculation of the average.

(2012) as a reference. We observe that sBOA* can be an order-of-magnitude faster than NAMOA*, and NAMOA*dr can be an order-of-magnitude faster than sBOA*. BOA* can be several times faster than NAMOA*dr, especially on instances with large numbers of Pareto-optimal solutions. For example, BOA* is 4.3 times faster than NAMOA*dr on FL (with 739 Pareto-optimal solutions on average), while BOA* is only 1.3 times faster than NAMOA*dr on BAY (with 119 Pareto-optimal solutions on average). BOA* can also be an order-of-magnitude faster than BBDijkstra and BDijkstra.

We now compare the runtimes of the two fastest algorithms, BOA* and NAMOA*dr, as a function of the difficulty of the instances on a large road map, namely the Great Lakes (LKS) map with 2,758,119 states and 6,885,658 edges. We include BDijkstra in the experiment. Figure 2 shows the runtimes (in seconds) of BOA*, NAMOA*dr, and BDijkstra on the 74 instances from Sedeño-Noda and Colebrook (2019) that BDijkstra solves within a runtime limit of 3,600 seconds. The instances are ordered in increasing numbers of their Pareto-optimal solutions ($|sols|$). When $|sols|$ is small, the runtimes of the algorithms are similar. When $|sols|$ increases, the runtimes of the algorithms increase. The runtime of BOA* increases smoothly and becomes orders of magnitude smaller than the ones of NAMOA*dr and BDijkstra. Figure 3 provides a different view of the results. It shows the cumulative runtimes (in seconds) of BOA*, NAMOA*dr, and BDijkstra. The instances are ordered in increasing runtime of BOA*. For instances where BOA* has small cumulative runtimes, the cumulative runtimes of the algorithms are similar. When the cumulative runtime of BOA* increases, it increases less than the ones of NAMOA*dr and BDijkstra.

We now compare the number of op-pruning operations[3] of BOA* and NAMOA*dr for the dominance checks on $\mathbf{G}_{op}$. Table 3 shows the number of Pareto-optimal solutions, the ratio of generated nodes and the ratio of runtimes of NAMOA*dr and BOA*, and the number of op-pruning operations per generated node for NAMOA*dr on four LKS instances. BOA* generates around 1.04 times more nodes than NAMOA*dr. For Instance 1, NAMOA*dr and BOA* run about equally fast. However, for the other instances, BOA* runs faster because NAMOA*dr performs the more op-pruning operations the larger $|sols|$ is, which demonstrates the advantage of BOA*, whose dominance checks run in constant time, over NAMOA*dr, whose dominance checks on $\mathbf{G}_{op}$ run only in linear time.

[3] so named by PMP: the number of nodes checked on $\mathbf{G}_{op}$ when a node is generated.

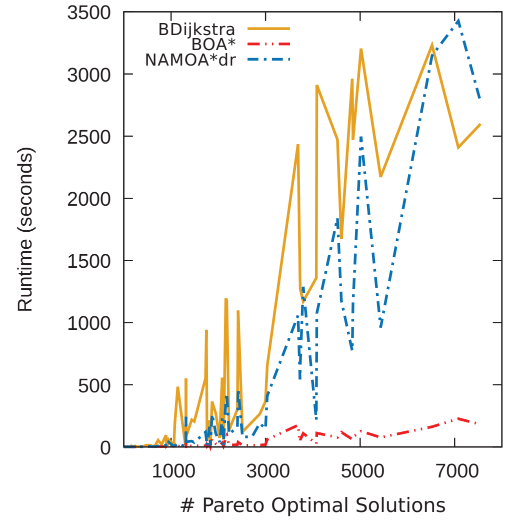

Figure 2: Runtime on 74 LKS instances. The instances on the x-axis are ordered in increasing numbers of their Pareto-optimal solutions.

We now determine the runtime of BOA* as a function of the lexicographic ordering used for the $Open$ list, namely either (f_1, f_2) or (f_2, f_1). Table 4 shows the runtime (in seconds) of BOA* with both the (f_1, f_2) and (f_2, f_1) orderings of the $Open$ list on 100 LKS instances. BOA* is faster when its $Open$ list is ordered lexicographically according to (f_2, f_1) instead of (f_1, f_2). In particular, it solves 2 more instances and has smaller average, maximum, and minimum runtimes because it generates 10 percent fewer nodes (and, consequently, also performs fewer heap percolations). We conclude that the ordering of the cost components has a strongly influence on the runtime of BOA*.

Conclusions and Future Work

We have presented Bi-Objective A* (BOA*), a simple and fast best-first bi-objective search algorithm. BOA* improves the efficiency of the dominance checks substantially, which is key to improving the efficiency of the search. The dominance checks of BOA* require only constant time, while the ones of existing bi- and multi-objective search algorithms require linear time. Our experimental evaluation shows that BOA* can run an one order of magnitude (or more) faster than state-of-the-art algorithms such as NAMOA*, NAMOA*dr, Bi-Objective Dijkstra, and Bidirectional Bi-

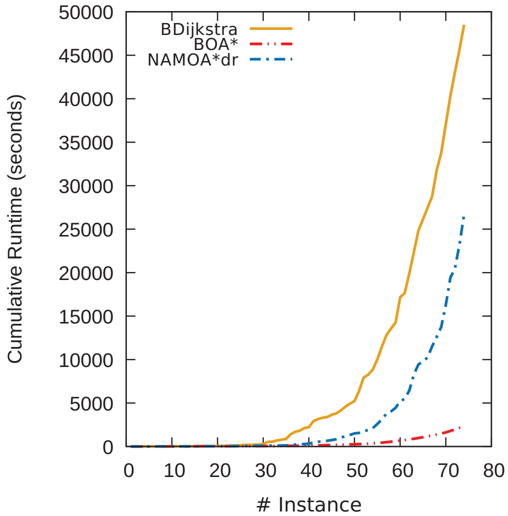

Figure 3: Cumulative runtime on 74 LKS instances. The instances on the x-axis are ordered in increasing runtime of BOA*.

Objective Dijkstra. We intend to improve and extend BOA* in future work as follows:

Speeding up BOA*: The cost of a solution is a pair (c_1, c_2). The c_1-values of solutions found by BOA* are strictly monotonically increasing in time, and the c_2-values are strictly monotonically decreasing in time. Thus, the first solution found by BOA* has the smallest c_1-value, and the last solution has the smallest c_2-value. If BOA* orders the $Open$ list lexicographically according to (f_2, f_1) instead of (f_1, f_2), the opposite happens. Thus, BOA* might run faster if it runs two BOA* instantiations in parallel, one for each ordering, and terminates when both instantiations find a solution of the same cost.

Selecting Solutions with BOA*: Several of our instances have thousands of Pareto-optimal solutions. For example, one of the LKS instances has 17,606 solutions. Many of the Pareto-optimal solutions are very similar in that they contain almost the same edges. We plan to extend BOA* so that it finds a subset of the cost-unique optimal solutions on the Pareto frontier that contains solutions that are sufficiently different from each other and thus good representatives of the Pareto frontier. Such an approach is especially beneficial when solutions need to be presented to human users for evaluation or selection.

Finding Bounded-Suboptimal Solutions: BOA* might be able to use weights, like Weighted A* (Pohl 1970), to obtain the Pareto frontier of all bounded-suboptimal solutions rather than the one of all optimal solutions. Our preliminary experiments show an impressive speed-up when weight $w = 1.2$ is used in the calculation of f_1. For example, BOA* found 3,686 optimal solutions in 175 seconds for the LKS instance with start state 2,258,596 and goal state 2,042,316, and Weighted BOA* found 4,023 solutions in 2.3 seconds. Our main challenge is to prove that the solutions set found by Weighted BOA* contains exactly all cost-unique w-suboptimal solutions on the Pareto frontier.

Using More Than Two Objective Functions: BOA* might be able to find all cost-unique Pareto-optimal solutions for cost functions with more than two components if it runs several times for different permutations of the components. For example, BOA* might find a subset of the Pareto-optimal solutions if it orders the $Open$ list lexicographically according to some ordering of the components. Other orderings might result in different subsets. Our main challenge is to prove that the union of all such subsets contains exactly all cost-unique optimal solutions on the Pareto frontier.

Acknowledgments

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, and 1935712. The research at Washington University in St. Louis was supported by NSF under grant numbers 1812619 and 1838364. Jorge Baier was supported by a FondDCC grant from the Dept. of Computer Science at PUC Chile.

References

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.

Edelkamp, S., and Schrödl, S. 2011. *Heuristic Search: Theory and Applications*. Morgan Kaufmann.

Hart, P. E.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Machuca, E., and Mandow, L. 2012. Multiobjective heuristic search in road maps. *Expert Systems with Applications* 39(7):6435–6445.

Mandow, L., and Pérez-de-la-Cruz, J. 2010. Multiobjective A* search with consistent heuristics. *Journal of the ACM* 57(5):27:1–27:25.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3):193–204.

Pulido, F.-J.; Mandow, L.; and Pérez-de-la-Cruz, J.-L. 2015. Dimensionality reduction in multiobjective shortest path search. *Computers & Operations Research* 64:60–70.

Raith, A., and Ehrgott, M. 2009. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research* 36(4):1299–1331.

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

Stewart, B. S., and White III, C. C. 1991. Multiobjective A*. *Journal of the ACM* 38(4):775–814.