Bounded-Cost Bi-Objective Heuristic Search

Shawn Skyler[1], Dor Atzmon[1], Ariel Felner[1], Oren Salzman[2]
Han Zhang[3], Sven Koenig[3], William Yeoh[4], Carlos Hernández Ulloa[5]
[1] Ben-Gurion University of the Negev
[2] Technion - Israel Institute of Technology
[3] University of Southern California
[4] Washington University in St. Louis
[5] Universidad Andrés Bello
shawn@post.bgu.ac.il, dorat@post.bgu.ac.il, felner@bgu.ac.il, osalzman@cs.technion.ac.il
zhan645@usc.edu, skoenig@usc.edu, wyeoh@wustl.edu, carlos.hernandez@uss.cl

Abstract

There are many settings that extend the basic shortest-path search problem. In *Bounded-Cost Search*, we are given a constant bound, and the task is to find a solution within the bound. In *Bi-Objective Search*, each edge is associated with two costs (objectives), and the task is to minimize both objectives. In this paper, we combine both settings into a new setting of Bounded-Cost Bi-Objective Search. We are given two bounds, one for each objective, and the task is to find a solution within these bounds. We provide a scheme for normalizing the two objectives, introduce several algorithms for this new setting and compare them experimentally.

Introduction and Background

A* (Hart, Nilsson, and Raphael 1968) and its many variants are used to solve classical shortest-path problems optimally. Nodes n are expanded according to $f(n) = g(n) + h(n)$. If $h(n)$ is *admissible*, an optimal solution path will be found. Nevertheless, there are other settings that do not require optimal solution paths. One such setting is *Bounded-Cost Search* (BCS) (Stern et al. 2014). In BCS, we are given a bound C, and the task is to quickly find a solution path with cost $\leq C$.

Another setting is *Bi-Objective Search* (BOS) (Raith and Ehrgott 2009). In BOS, we are given a graph (state-space) where two types of costs are associated with its edges that need to be minimized, e.g., travel distance and time for transportation problems. BOS has many applications, ranging from robotics (Fu et al. 2019; Fu, Salzman, and Alterovitz 2021) to transportation (Bronfman et al. 2015). The notion of an optimal path does not exist here. Instead, the "best" possible paths are those that are *undominanted* by other paths. Path π is said to *dominate* path π' iff both of π's costs are not larger than the corresponding costs of π' and at least one cost of π is strictly smaller than the corresponding cost of π'. The *Pareto-Optimal Frontier* (POF) is the set of solution paths that are undominated by other solution paths. The BOA* algorithm (Hernández Ulloa et al. 2020) is considered the state-of-the-art algorithm for finding the POF.

In this paper we combine both BCS and BOS to define the problem of Bounded-Cost Bi-Objective Search (BC-BOS) where we are given a pair of bounds, one for each of the two costs, and need to find a solution path whose costs are below these bounds. In particular, in this paper, we are interested in the more restrictive problem of finding a solution path in the POF whose costs are below the bounds.

We first provide a normalization mechanism that converts the costs of both objectives (which could have different scales, e.g., time and distance) to values in the range of $[0 \dots 1]$. This places the two objectives on the same scale and, therefore, makes them directly comparable. We then introduce the BCP-BOA* algorithm, which modifies BOA* to return a POF solution path within the given bounds. BCP-BOA* uses our new notion of *bound pruning*, which prunes nodes that are not within the bounds. We introduce several variants of BCP-BOA* that explore the set of required nodes by using various ordering functions to order nodes in *OPEN*. We then experimentally compare the variants of BCP-BOA* on different pairs of cost bounds and study the advantages and disadvantages of each algorithm. Indeed, all variants run much faster than BOA*, that generates the full set of POF solution paths.

Definitions and Background

A *Bi-Objective Search* (BOS) problem is characterized by a graph $G = (V, E)$, where V is a set of vertices (states) and E is a set of edges, a start vertex $start \in V$ and a goal vertex $goal \in V$. We use a **boldface** font to indicate a pair of two numbers, e.g., $\mathbf{b} = (b_1, b_2)$. The addition of two pairs $\mathbf{x}$ and $\mathbf{y}$ yields $(x_1 + y_1, x_2 + y_2)$. We say that $\mathbf{x}$ *dominates* $\mathbf{y}$ ($\mathbf{x} \prec \mathbf{y}$) iff $(x_1 \leq y_1 \wedge x_2 < y_2)$ or $(x_1 < y_1 \wedge x_2 \leq y_2)$. Each edge $e \in E$ is associated with two cost functions $\mathbf{c}(e) = (c_1(e), c_2(e))$, i.e., a cost for each objective. A path $\pi = [v_1, \dots, v_n]$ is a list of neighboring vertices. The cost of path π is $\mathbf{C}(\pi) = (C_1(\pi), C_2(\pi)) = \sum_{i=1}^{n-1} \mathbf{c}(v_i, v_{i+1})$.

A solution path π is a path from $start$ to $goal$. Since there are two costs in BOS, the notion of an optimal solution path does not exist. Instead, the *Pareto-Optimal Frontier* (POF) is the set of solution paths Π such that each path $\pi \in \Pi$ is not *dominated* by any other solution path. The points in Figure 1(a) show the POF costs where the x-axis represents c_1 and the y-axis represents c_2.

For BOS, an *admissible* heuristic function $\mathbf{h}(n) =$

$(h_1(n), h_2(n))$ is a lower bound on the costs of a path from a given node n to $goal$. We denote the heuristic for the node of the start vertex by $\mathbf{h}(start)$ (resp. $\mathbf{h}(goal)$ for the goal vertex). We also assume that $\mathbf{h}$ is *consistent* (i.e., $\forall_{i \wedge e \in E}$: $h_i(goal) = \mathbf{0} \wedge h_i(n) \leq c_i(e = (n, n')) + h_i(n')$). Specifically, we follow the common practice in BOS and use the *individual shortest path* heuristic function (Hernández Ulloa et al. 2020; Goldin and Salzman 2021; Pulido, Mandow, and Pérez-de-la Cruz 2015). That is, for $\mathbf{h}(n)$, $h_i(n)$ is the cost-minimal path from n to $goal$ using the ith objective only. For example, consider a problem instance with only two solution paths with cost $(1, 10)$ and $(11, 2)$. Then, $\mathbf{h}(start) = (1, 2)$. Computing such cost-minimal paths can be done in time polynomial in V (the number of vertices of the input graph). This can be neglected compared to the complexity of BOS, which is exponential in V. In fact, we can assume that the *all-pairs-shortest-path* (cost-minimal) data for each of the objectives is given as input along with the graph, e.g., by running a polynomial time preprocessing phase.

Previous Work on BOS

In BOS, the number of nodes can be exponentially larger than the number of vertices of the underlying graph because every path to a vertex has its unique node with possibly unique f-values. In fact, the POF itself can contain an exponential number of paths (Ehrgott 2005; Breugem, Dollevoet, and Heuvel 2017). Several A*-based search algorithms have been designed to find the POF. Examples include Multi-Objective A* (MOA*) (Stewart and White III 1991), NAMOA* (Mandow and De La Cruz 2005) and NAMOA*-dr (Pulido, Mandow, and Pérez-de-la Cruz 2015). All these algorithms use the same general *best-first search framework*, denoted here by BO-BFS. BO-BFS employs the traditional *OPEN* and *CLOSED* lists, and chooses a node $n \in$ *OPEN* for expansion that is undominated by other nodes in *OPEN*. A common practice is to order nodes according to their *lexicographic order*, i.e., first compare f_1 and, in a case of a tie, compare f_2. BO-BFS performs *dominance checks* on newly generated nodes to prune nodes that are dominated by other nodes that have already been generated or expanded. These dominance checks usually incur CPU overhead which is linear in the size of *OPEN*. The BO-BFS algorithms mainly differ in low-level implementation details of the expansion cycle and the dominance checks.

BOA* (Hernández Ulloa et al. 2020) is a BO-BFS algorithm, which is considered the current state-of-the-art. BOA* exploits the fact that nodes are ordered in lexicographic order of their f-values to perform all dominance checks in constant time. Unlike the other algorithms, when a node n is generated, BOA* compares its g_2-value (the g-value of the second objective) only against the best seen g_2-value for vertex v of node n. Additionally, BOA* compares $f(n)$ with the cost of the best known solution path.

Bounded-Cost BOS

The *Bounded-Cost Bi-Objective Search* (BC-BOS) problem is a BOS problem which is also characterized by a pair of bounds (budgets) $\mathbf{b} = (b_1, b_2)$. A solution path is now a path $\pi = [start, \ldots, goal]$ whose costs are within the bounds, i.e., $\mathbf{C}(\pi) \preceq \mathbf{b}$. The task is to find a solution path as fast as possible. In this paper, we focus on the more restrictive problem of finding a solution path that is also in the POF.[1] We call this restricted problem BCP-BOS to differentiate it from the general BC-BOS problem that seeks a solution path which may or may not be in the POF.

Normalizing the Two Objectives

In many problem instances, the two objectives may not be measured on the same scale, e.g., time in minutes and distance in miles. Therefore, we introduce a normalizing procedure that maps all values into the range $[0 \ldots 1]$, which allows us to compare and combine both objectives since they are now on the same scale. We use this normalization method in this paper, but it is general and can be used by other BOS algorithms too.

Extreme Costs of Solution Paths

Let $\min_1 = h_1(start)$ (the c_1-cost of the cost-minimal path from $start$ to $goal$), and let OPT_1 be the set of all solution paths with cost $\min_1$. Now, let $\max_2 = \min_{\pi \in \text{OPT}_1} C_2(\pi)$ (namely, $\max_2$ is the minimal c_2-cost of paths whose c_1-cost is $\min_1$). $\min_2$, OPT_2 and $\max_1$ are defined analogously. Note that $\min_i$ can be computed in time $O(E)$ by running a Single-Objective A* search with only the costs of the ith objective, and breaking ties according to the minimal cost of the other objective. As we use the individual shortest path heuristic functions for each objective, this computation is extremely fast.

$(\min_1, \max_2)$ and $(\max_1, \min_2)$ are the most extreme solution costs in the POF (see the top-left and bottom-right points in Figure 1(a)), and $\mathbf{h}(start) = (\min_1, \min_2)$. For example, assume that the two extreme POF solution paths have costs $(1, 10)$ and $(11, 2)$. Then $\min_1 = 1$, $\max_1 = 11$, $\min_2 = 2$ and $\max_2 = 10$. Naturally, the costs of any other POF solution paths are between these values.

Given $\min_i$ and $\max_i$, we can easily solve the BCP-BOS problem in the following cases: (i) If one of the bounds satisfies $b_i < \min_i$ then no solution path exists. (ii) If one of the bounds satisfies $b_i > \max_i$ then one of the extreme solution paths in the POF (within the bounds) is a valid solution. Thus, henceforth, we limit our discussion to the setting where $\min_i \leq b_i \leq \max_i$ for both $i \in \{1, 2\}$.

Normalizing the Costs

We now define a normalization function for any cost value x_i with $\min_i \leq x_i \leq \max_i$ (for $i \in \{1, 2\}$). We define:

$$\bar{x_i} = \frac{x_i - \min_i}{\max_i - \min_i}$$

This normalization maps all cost values into the interval $[0 \ldots 1]$ (visualized in Figure 1(b)). Hence, the two (now normalized) cost functions can be compared and combined. Moreover, the extreme solution paths in the POF now have

[1] In classical single-objective search, there is only one optimal solution cost, so finding an optimal solution path below a given bound makes no sense. One must run A* and find an optimal solution path.

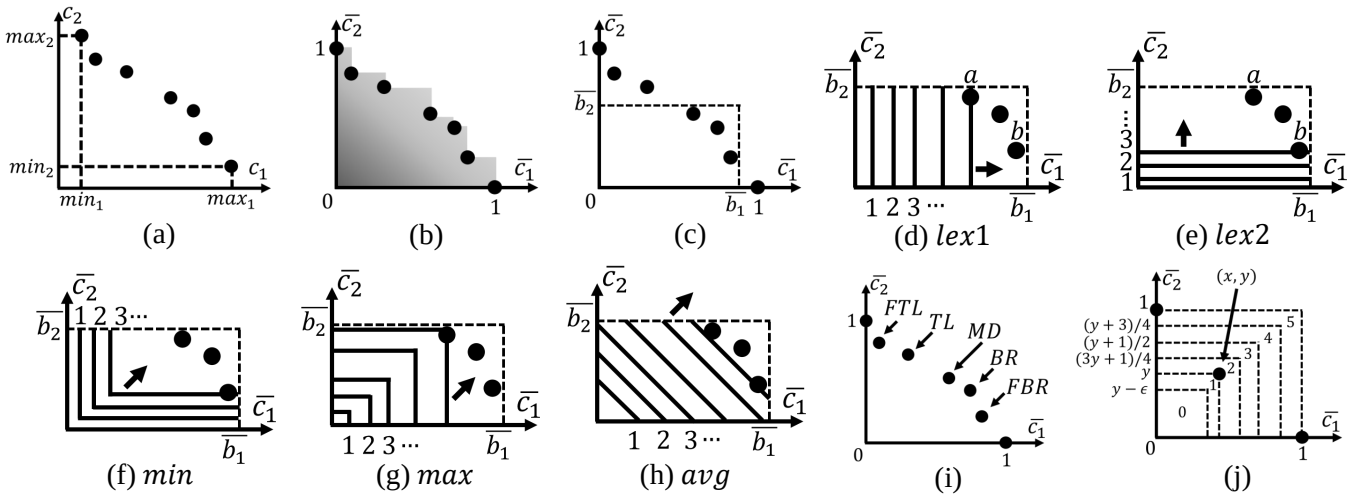

Figure 1: Bi-objective search illustration.

normalized costs $(0, 1)$ and $(1, 0)$, respectively (see the top-left and bottom-right points in Figure 1(b)).

An Algorithm for Solving BCP-BOS

We now present BCP-BOA*, an algorithm that solves BCP-BOS. BCP-BOA* uses the BO-BFS framework described above but modifies it to fit BCP-BOS.

BCP-BOA* executes a best-first search, which maintains an *OPEN* list and a *CLOSED* list. *OPEN* contains all leaf nodes of the current (partial) search tree, and *CLOSED* contains all already-expanded non-leaf nodes of the tree. Each node contains a vertex v, a $\mathbf{g}$-value (g_1, g_2) (with the cost of reaching vertex v from $start$) and an $\mathbf{h}$-value (h_1, h_2) (with a consistent estimate of cost of reaching $goal$ from v). As described above, we use the individual shortest path heuristic function.

BCP-BOA* starts by inserting a root node r into *OPEN* that contains the start vertex $start$ and $\mathbf{f}(r) = \mathbf{g}(r)+\mathbf{h}(r) = (0, 0) + (\min_1, \min_2) = (\min_1, \min_2)$ (= $(0, 0)$ after normalizing the values). In each iteration, BCP-BOA* chooses the *best* node for expansion and expands it. Below, we discuss different methods for choosing which node to expand next. When a node n that contains vertex v is expanded, BCP-BOA* moves n from *OPEN* to *CLOSED* and creates a new node n' for each successor vertex v' of v with $\mathbf{g}(n') = \mathbf{g}(n) + \mathbf{c}((v, v'))$. Next, to address the bounded-cost requirement for each new node n', BCP-BOA* discards n' if one of its f_i-values exceeds the bounds (i.e., if $f_1(n') = g_1(n') + h_1(n') > b_1$ or if $f_2(n') = g_2(n') + h_2(n') > b_2$). This step is called *bound pruning*. If both costs are within their bounds, BCP-BOA* performs a *dominance check*. That is, BCP-BOA* checks if there is a node $n'' \in$ *OPEN* $\cup$ *CLOSED* with $\mathbf{g}(n'') \preceq \mathbf{g}(n')$ that contains the same vertex as n'. In this case, it prunes n'. Otherwise, it inserts n' into *OPEN*. The search terminates when a node that contains the goal vertex is expanded. Other BO-BFS algorithms might perform the dominance checks differently, but the ideas behind BCP-BOA* apply to them as well.

All BO-BFS algorithms mentioned above return the *entire* POF (by expanding all nodes in the grey area in Figure 1(b)). In contrast, BCP-BOA* uses bound pruning to expand nodes only in the rectangle in Figure 1(c), which is bounded by the two bounds. BCP-BOA* can be tuned to continue the search with bound pruning until *OPEN* is empty. In this case, it will find all solutions that are both in the POF and the rectangle.

All BO-BFS algorithms mentioned above store $f_1(n)$ and $f_2(n)$ for each node $n \in$ *OPEN*. They typically order *OPEN* in lexicographic order of the $\mathbf{f}$-values of the nodes. That is, the best node in *OPEN* is the node with the smallest f_1-value, and the smallest f_2-value among all nodes with the smallest f_1-value. This allows BOA* to perform dominance checks in constant time (Hernández Ulloa et al. 2020). The implication of using this lexicographic ordering is that solution paths are found from the top-left of the POF to its bottom-right. However, since we are only interested in finding one solution path, BCP-BOA* has more flexibility in how it orders the nodes in *OPEN*. We describe several approaches for this purpose next.

Choosing the Best Node in *OPEN*

Given the normalized bounds $\bar{b_1}$ and $\bar{b_2}$, BCP-BOA* has to search the rectangle whose extreme points are (0,0) and $(\bar{b_1}, \bar{b_2})$, as illustrated in Figure 1(c). If a solution path exists, the POF points cross this rectangle diagonally (see again Figure 1(c)). We now describe strategies that search this rectangle systematically to find a solution path as fast as possible (see Figures 1(d-h) for visualizations). We define such strategies using an *ordering function*. A general ordering function O maps two single objective functions F_1 and F_2, and two nodes n and m to the suggested node for expansion among n and m. It creates a total order between all nodes, thereby determining the order of the nodes in *OPEN*. O is defined as follows:

$$O(F_1, F_2, n, m) = \begin{cases} n \ if & F_1(n) < F_1(m) \\ n \ if & (F_1(n) = F_1(m)) \wedge \\ & (F_2(n) < F_2(m)) \\ m & \text{otherwise.} \end{cases}$$

We next describe a number of ordering functions that are obtained from the general ordering function by defining F_1 and F_2 differently.

Lexicographic Orderings

We define the $Lex1$ and $Lex2$ ordering functions as follows:

$$O_{\text{lex1}}(n, m) = O(f_1, f_2, n, m)$$
$$O_{\text{lex2}}(n, m) = O(f_2, f_1, n, m)$$

Their orders of node expansions are shown in Figures 1(d,e). O_{lex1} first expands nodes along the left vertical line from bottom to top, then the second left vertical line from bottom to top, and so on. O_{lex2} first expands nodes along the bottom horizontal line from left to right, then the nodes along the second horizontal line from the bottom, and so on. In the figure, $\bar{b_1}$ is larger than $\bar{b_2}$. Since the POF crosses the top-right part of the rectangle defined by the bounds diagonally, O_{lex2} has an advantage over O_{lex1} as it expands a node in the POF earlier (node b in Figure 1(e)) after scanning less than half of the rectangle. In contrast, O_{lex1} scans the rectangle from left to right and needs to scan more than half of the rectangle before expanding node a in the POF. Therefore, when $\bar{b_1}$ is larger than $\bar{b_2}$, O_{lex2} often outperforms O_{lex1} and vice versa.

Minimum and Maximum Orderings

Let $F_{\min}(n)$ and $F_{\max}(n)$ be two functions returning the minimal and maximal normalized f-values of a node n, respectively. Namely, $F_{\min}(n) = \min(\bar{f_1}(n), \bar{f_2}(n))$ and $F_{\max}(n) = \max(\bar{f_1}(n), \bar{f_2}(n))$. We define the Min and Max ordering functions (see Figures 1(f,g)) as follows:

$$O_{\min}(n, m) = O(F_{\min}, F_{\max}, n, m)$$
$$O_{\max}(n, m) = O(F_{\max}, F_{\min}, n, m)$$

Min combines $Lex1$ and $Lex2$, while Max is similar to a depth-first search.

Average Ordering

Let $F_{\text{avg}}(n)$ be the average normalized f-values of a node. Namely, $F_{\text{avg}}(n) = (\bar{f_1}(n) + \bar{f_2}(n))/2$. We define the Average ordering function (visualized in Figure 1(h)) as follows:

$$O_{\text{avg}}(n, m) = (F_{\text{avg}}, F_{\min}, n, m)$$

Ties are broken according to the minimal f-value of each node. We also tried other ordering functions, such as weighted average, circle-shaped $\bar{f_1}(n)^2 + \bar{f_2}(n)^2$ and hyperbole-shaped $\bar{f_1}(n) \cdot \bar{f_2}(n)$ ordering functions. We do not report them as we did not observe any important benefits for them.

Experimental Results

We compared all variants of BCP-BOA* on the BAY road map[2]. A common practice is to use time and distance as objectives on this map (Hernández Ulloa et al. 2020). For the purpose of the experiments, we first found the POF for each problem instance. We then chose five nodes from the POF as *pivots* to set the two bounds between $\min_i$ and $\max_i$ ($i \in \{1, 2\}$). The pivots selected are (see Figure 1(i)): **(1) FTL**, which is the farthest top-left POF solution path that is not an extreme solution path; **(2) FBR**, which is the farthest bottom-right POF solution path that is not an extreme solution path (analog to FTL); **(3) MD**, which is the POF solution path that has the smallest difference between its $\bar{c_1}$ and $\bar{c_2}$ costs; **(4) TL**, which is the POF solution path that is closest to the middle between MD and FTL; and **(5) BR**, which is the POF solution path that is closest to the middle between MD and FBR.

For a given pivot with costs $\bar{\mathbf{C}} = (x, y)$, we divide the space of solution paths into six zones, as shown in Figure 1(j). In Zone 0, we set $\bar{\mathbf{b}} = (x - \epsilon, y - \epsilon)$, and therefore there are no solution paths in that zone. In Zone 1, we set $\bar{\mathbf{b}} = (x, y)$, and the cost of the path to the pivot is the only possible cost. In Zone 5, we set $\bar{\mathbf{b}} = (1, 1)$, and it contains all POF solution paths. The most interesting zones are 2, 3 and 4, which divide the area between (x, y) and $(1, 1)$ into three zones (that all include at least one but not all POF solutions). For these zones, the bounds are $\bar{\mathbf{b}} = (\frac{1}{4}(3x+1), \frac{1}{4}(3y+1))$, $\bar{\mathbf{b}} = (\frac{1}{2}(x+1), \frac{1}{2}(y+1))$, and $\bar{\mathbf{b}} = (\frac{1}{4}(x+3), \frac{1}{4}(y+3))$, respectively.

Table 1 presents the average number of node expansions over 135 problem instances for Zones 2, 3, 4 and 5. For Zones 2–4, we report numbers for the five pivots defined above. The first two rows present the average bounds. Then, each row represents a different ordering function. For Zone 5, we only report one number. Zone 5 always contains all nodes in the POF and is thus similar for all pivots. Zones 0 and 1 are not included since all variants of BCP-BOA* need to expand the same number of nodes in these zones. In general, given two nodes $n_1 \prec n_2$, BCP-BOA* with all ordering methods expands n_1 before n_2. Thus, when there is no solution path or when only a single solution path with cost $\bar{\mathbf{b}}$ exists (Zones 0 and 1), all variants expand the same number of nodes.

In Zones 2–4, $Lex1$ was almost always better than $Lex2$ in FTL and TL, while $Lex2$ was better than $Lex1$ in MD, BR and FBR. The advantage of $Lex2$ for MD is probably due to some skewing of the exact map and the order of the parameters (time and distance). *Selective Lex* is an intelligent variant which selects $Lex2$ if $\bar{b_1} > \bar{b_2}$ and $Lex1$ otherwise. Thus, it exploits the benefits of both Lex variants. Our results confirm that *Selective Lex* was the most robust variant: it was either the best ordering function or very close to it. Min also performed relatively well and was sometimes the best. The remaining ordering functions performed poorly.

We do no report runtimes because the runtimes per node expansion were very similar for all ordering functions. BCP-BOA* with the Lex ordering functions can exploit

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

	Zone 2					Zone 3					Zone 4					Zone 5
Ordering	FTL	TL	MD	BR	FBR	FTL	TL	MD	BR	FBR	FTL	TL	MD	BR	FBR	Any
$\bar{b}_1$	.26	.42	.51	.60	.96	.50	.61	.67	.73	.97	.74	.80	.83	.85	.98	1.00
$\bar{b}_2$	.95	.60	.51	.43	.21	.97	.74	.76	.62	.50	.98	.87	.84	.81	.75	1.00
Lex1	**1.1**	**83.6**	105.8	110.3	81.6	**1.1**	61.1	88.2	107.6	106.9	**1.1**	26.7	52.6	55.8	84.3	1.1
Lex2	117.7	107.4	**78.3**	**49.1**	**0.8**	103.7	68.5	**44.3**	**24.6**	**0.8**	38.5	19.5	**11.3**	**6.3**	**0.8**	**0.8**
Selective Lex	**1.1**	**83.6**	92.8	**49.1**	**0.8**	**1.1**	61.1	59.7	**24.6**	**0.8**	**1.1**	26.6	16.5	**6.3**	**0.8**	**0.8**
Min	1.5	96.6	107.2	65.3	0.9	1.8	**60.4**	71.0	36.3	1.0	2.1	**18.7**	15.3	9.5	1.2	1.3
Max	117.8	123.8	124.5	120.5	87.9	123.7	124.0	124.5	124.2	120.6	124.5	124.5	124.5	124.5	124.5	124.4
Average	141.6	174.7	175.4	161.1	113.6	173.6	180.7	183.4	179.1	166.0	181.0	182.1	181.3	181.3	182.1	181.3
All POF	162.8	226.0	241.5	245.9	202.8	264.4	303.9	312.4	314.0	306.1	337.6	349.2	353.6	356.3	354.0	369.4
#POF Solutions	67	56	56	51	50	98	92	88	84	86	124	120	118	115	116	147

Table 1: Number of expanded nodes (in thousands) for Normalized BCP-BOA*.

the constant-time dominance checks of BOA*. While BCP-BOA* with the other ordering functions has only linear-time dominance checks, they are linear in the number of nodes that contain the same vertex. Since we used bound pruning, that number was very small and did not affect the runtimes.

When BCP-BOA* keeps running after it finds the first POF solution path, it finds *all* possible POF solution paths within the given bounds. The number of node expansions for this continuous variant of BCP-BOA* is also presented (ALL POF) as well as the number of solution paths that it found. On average, there were 147 different POF solution paths, and 369K node expansions were needed to find them. In contrast, for any given bounds, BCP-BOA* with the best ordering function reduced the number of node expansions by factors from 2.7 (Zone 2, TL) to over 400 (Zone 4, FBR) when finding a single POF solution path.

Conclusions

We presented BCP-BOA* and several variants of it, and concluded that *Selective Lex* is the best ordering function for ordering nodes in *OPEN*. It is future work to lift the requirement that the returned solution path must be in the POF, thereby allowing variants of Potential Search (Stern et al. 2014) to be used. Our normalization scheme can be adapted to other BOS algorithms. Finally, our work can be generalized to *Multi-Objective Search*.

Acknowledgements

This research was supported by the United States-Israel Binational Science Foundation (BSF) under grant numbers 2017692 and 2021643, Israel Science Foundation (ISF) under grant number 844/17, the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, 2112533, 2121028, and by Centro Nacional de Inteligencia Artificial under grant number FB210017. 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 the U.S. government.

References

Breugem, T.; Dollevoet, T.; and Heuvel, W. 2017. Analysis of FPTASes for the Multi-Objective Shortest Path Problem. *Computers & Operations Research*, 78: 44–58.

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.

Ehrgott, M. 2005. *Multicriteria Optimization*, volume 491. Springer Science & Business Media.

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 XV*.

Fu, M.; Salzman, O.; and Alterovitz, R. 2021. Computationally-Efficient Roadmap-Based Inspection Planning via Incremental Lazy Search. In *ICRA*, 7449–7456.

Goldin, B.; and Salzman, O. 2021. Approximate Bi-Criteria Search by Efficient Representation of Subsets of the Pareto-Optimal Frontier. In *ICAPS*, 149–158.

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

Hernández Ulloa, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; and Koenig, S. 2020. A Simple and Fast Bi-Objective Search Algorithm. In *ICAPS*, 143–151.

Mandow, L.; and De La Cruz, J. L. P. 2005. A New Approach to Multi-Objective A* Search. In *IJCAI*, 218–223.

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 Cf Solution Strategies for Biobjective Shortest Path Problems. *Computers & Operations Research*, 36(4): 1299–1331.

Stern, R.; Felner, A.; van den Berg, J.; Puzis, R.; Shah, R.; and Goldberg, K. 2014. Potential-Based Bounded-Cost Search and Anytime Non-Parametric A$*$. *Artificial Intelligence*, 214: 1–25.

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