Bounded Cost Bi-Objective Heuristic Search

SoCS-2022 Submission #46

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 these 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.

1 Introduction and Background

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

Another search or pathfinding setting is *Bi-Objective Search* (BOS) (Raith and Ehrgott 2009). In this setting, the underlying state-space (graph) has two types of costs associated with its edges that need to be minimized, e.g., travel distance and time for transportation problems. The BOS setting has diverse applications ranging from robotics (Fu et al. 2019; Fu, Salzman, and Alterovitz 2021) to transportation problems (Bronfman et al. 2015). The notion of an optimal path does not exist here. Instead, the "best" possible paths are those who are un-dominanted. Path π is said to *dominate* path π' if both of π's costs are not larger than those 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, each of which is not dominated by any other solution path.

In BOS, the number of nodes may be exponentially larger than the number of states of the underlying graph because every path to a state has its unique node and the POF itself may contain an exponential number of paths (Ehrgott 2005; Breugem, Dollevoet, and Heuvel 2017). Several algorithms that are based on A^* were designed to find the POF. Examples include the 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). They all expand nodes according to their f-values using the traditional *OPEN* and *CLOSED* lists. All these algorithms perform *dominance checks* on newly generated nodes in order to prune away nodes that are dominated by other nodes that were generated or expanded. These dominance checks usually incur CPU overhead which is linear in the size of *OPEN*. BOA^* (Hernandez et al. 2020), which is considered the current state-of-the-art, exploits the fact that nodes are ordered in lexicographical-order and performs all dominance checks in constant time. Unlike the former algorithms, when a node n is generated BOA^* compares the current g_2-value (the g-value of the second objective) only against the best seen g_2-value of n. Additionally, BOA^* compares $f(n)$ with the best relevant known solution.

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 path whose costs are below these bounds. In particular, we are interested in a POF solution whose costs are below the bounds.[1] To efficiently solve BC-BOS, we introduce a new notion of pruning which we call *bound pruning*. More importantly, we characterize the set of nodes a best-first search algorithm needs to explore and suggest several natural methods to systematically explore this set of nodes. This requires combining the f-values of the two objectives into a single priority value—a unique challenge that does not appear in the general BOS problem. Thus, we provide a normalizing mechanism that converts the cost values of both objectives (which could have different scales, e.g., time and distance) to values in the range of $[0 \dots 1]$.

Given these algorithmic tools, (bound pruning, normalization and single-priority functions), we introduce the BC-BOA^* algorithm, which modifies BOA^* to return a POF

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

solution within the given bounds. We then experimentally compare several variants of BC-BOA* on different pairs of cost-bounds and study the pros and cons of each method. Indeed, all these variants run much faster than the full BOA* which generated the full set of POF solutions.

2 Problem Definition

The input to a *Bi-objective Search* (BOS) problem is a graph $G = (V, E)$, where V is a set of vertices (states) and E is a set of edges, a start state $s \in V$, and a goal state $g \in V$. 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 $\mathbf{y}$ equals $(x_1 + y_1, x_2 + y_2)$. 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 $(x_1 < y_1 \wedge x_2 \leq y_2)$. Similarly, $\mathbf{x}$ *weakly dominates* $\mathbf{y}$ ($\mathbf{x} \preceq \mathbf{y}$) iff $(x_1 \leq y_1 \wedge x_2 \leq y_2)$. 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 of the objectives. A path $\pi = [s_1, \ldots, s_n]$ is a list of neighboring states. The cost of path π is $\mathbf{C}(\pi) = (C_1(\pi), C_2(\pi)) = \sum_{i=1}^{n-1} \mathbf{c}((s_i, s_{i+1}))$.

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

The *Bounded-Cost Bi-Objective Search* problem (BC-BOS) is a BOS problem which also receives a pair bounds (budgets) $\mathbf{b} = (b_1, b_2)$ as input. A solution to BC-BOS is a path $\pi = [s, \ldots, g]$ whose costs are within the budgets, i.e., $\mathbf{C}(\pi) \preceq \mathbf{b}$. The task in BC-BOS is to find a path within the bounds as fast as possible. In this paper, we focus on a more restricted problem which is to find a Bounded-cost path π that is also on the POF.

For BOS, an *admissible* heuristic $\mathbf{h}(n) = (h_1(n), h_2(n))$ is a lower-bound estimation of the cost of a path from a given state n to g. We also assume that $\mathbf{h}$ is *consistent* (i.e., $\forall_{i \wedge e \in E} : h_i(s_{\text{goal}}) = \mathbf{0} \wedge h_i(s) \leq c_i(e = \langle s, s' \rangle) + h_i(s')$). Specifically, we follow the common practice in BOS to use the *individual shortest path* heuristic function (see, e.g., (Hernandez et al. 2020; Goldin and Salzman 2021; Pulido, Mandow, and Pérez-de-la Cruz 2015)). That is, for $\mathbf{h}(n) = (h_1, h_2)$, $h_i(n)$ is the cost shortest path from n to g using the i'th objective only. For example, assuming a problem with only two solutions with cost $(1, 10)$ and $(11, 2)$, then $\mathbf{h}(s) = (1, 2)$. As the number of solutions may be exponential in the graph size, computing such distances (along a single objective) is assumed to be computationally-cheap when compared to the bi-objective problem.

3 Extreme Values of Solutions

In many problem instances the two objectives may not be on the same scale, e.g., time in minutes and distance in Miles. Therefore we introduce a general normalizing procedure that maps all values to be in the range of $[0 \ldots 1]$. This will allow us to compare and combine both objectives which will now be on the same scale.

Let $\min_1$ be the cost of the shortest path from s to g along the c_1-cost and let OPT_1 be the set of all paths in G with cost $\min_1$. Now, set $\max_2 := \min_{\pi \in \text{OPT}_1} c_2(\pi)$. 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 by running A* along the cost of the i'th objective and that OPT_i can be computed by continuing the last f-layer of the search until all solutions are found. As we assume to have a prefect heuristic (individual shortest path) along each objective, this execution is extremely fast. Furthermore, note that $\mathbf{h(s)} = (\min_1, \min_2)$ and that $(\min_1, \max_2)$ and $(\max_1, \min_2)$ are the most extreme solution costs on the POF (see the top-left and bottom-right circles in Figure 1(a)). For example, assuming two extreme POF solutions with 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 are between these costs.

Given the values $\min_i$ and $\max_i$, we can easily solve the BC-BOS problem in the following cases. (i) If one of the original bounds satisfies $b_i < \min_i$ then no solution exists. (ii) If one of the original bounds satisfies $b_i > \max_i$ then we set $b_i = \max_i$ because the c_i-cost of all solutions on the POF must be $\leq \max_i$. Thus, henceforth, we limit the discussion to the setting where $\min_i \leq b_i \leq \max_i$ for $i \in \{1, 2\}$.

3.1 Normalizing the Values

We now define a normalizing function for any value x_i (with $i \in \{1, 2\}$) where $\min_i \leq x_i \leq \max_i$. Specifically, we denote $\bar{x_i}$ to be the normalized value of x_i and set

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

This normalization maps all values to $[0 \ldots 1]$ (visualized in Figure 1(b)). Hence, the two (normalized) cost functions can be compared and combined. Moreover, the normalized extreme solutions on the POF (i.e., the top left and bottom right solutions) are now $(0, 1)$ and $(1, 0)$, respectively.

4 Algorithm for Solving BC-BOS

We now present BC-BOA*, an algorithm that solves BC-BOS. BC-BOA* modifies all the A*-based algorithms described above. BC-BOA* executes a best-first search, which maintains an *OPEN*- and *CLOSED*-lists. *OPEN* contains all the leaf nodes of the current partial search tree and *CLOSED* contains all the already-expanded non-leaf nodes of the tree. Each node contains a state n coupled with two $\mathbf{g}$-values ((g_1, g_2) – costs of reaching the state of node n from s) and two $\mathbf{h}$ values ((h_1, h_2) – admissible estimates on the remaining costs for reaching g). As described above we use the individual shortest path heuristic.

The search starts by inserting into *OPEN* a root node r that contains the start state s and $\mathbf{f}(r) = \mathbf{g}(r) + \mathbf{h}(r) = (0, 0) + (\min_1, \min_2) = (\min_1, \min_2)$. In each expansion cycle, the search chooses the *best* node for expansion and expands it. Below, we discuss different methods for choosing which node to expand next. When node n that contains state s is expanded, n is moved to *CLOSED* and we create a new node n' for each successor state $(s, s') \in E$ while setting $\mathbf{g}(n') \leftarrow \mathbf{g}(n) + \mathbf{c}(s, s')$. Next, to address the bounded-cost requirement then for each such new node n',

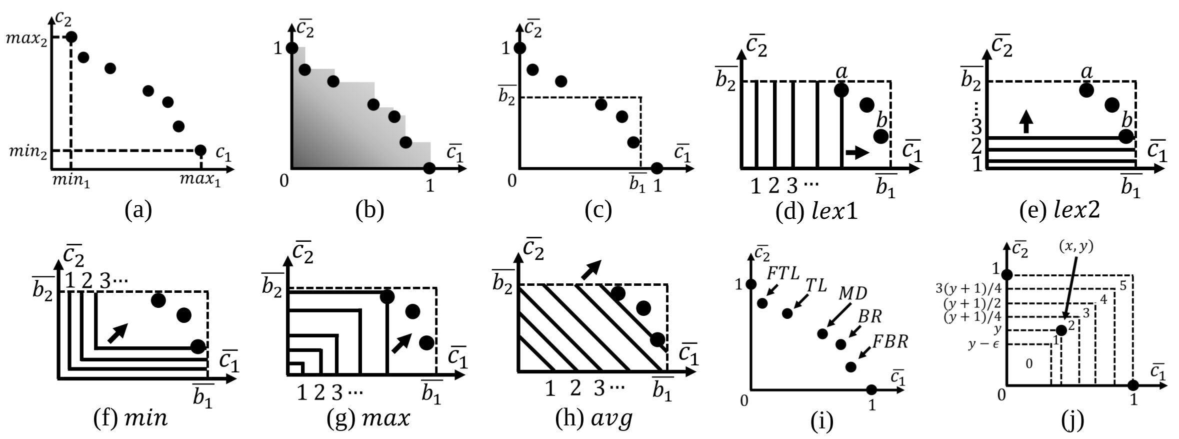

Figure 1: Bi-objective search illustration

if one of its f_i-values exceeds the budget (i.e., if $f_1(n') = g_1(n') + h_1(n') > b_1$ or if $f_2(n') = g_2(n') + h_2(n') > b_2$), n' is discarded. This step is called the *bound pruning* step. If both costs are within their budgets, we perform a *dominance check*. That is, we check if there is a node $n'' \in$ *OPEN* $\cup$ *CLOSED* that contains the same state with $\mathbf{g}(n'') \preceq \mathbf{g}(n')$. In this case, n' is pruned. Otherwise, n' is inserted into *OPEN*. Algorithms for returning the POF differ in how these dominance checks are done. But, BC-BOA* can work on top of all of them. The search halts when a node that contains the goal state is expanded. Pseudo code is provided in the supplementary material.

Importantly, the former algorithms (MOA*, NAMOA*, NAMOA*-dr and BOA*) and BC-BOA* return the *entire* POF (they expand all the nodes in the grey area in Figure 1(b)). By contrast, BC-BOA* activates the bound pruning step and therefore is constrained to only search within the rectangle depicted in Figure 1(c) which is bounded by the two bounds. We note that BC-BOA* can be tuned to find all the solutions that are both on the POF and are below the bounds if we keep on searching until *OPEN* is empty (while discarding nodes above the bound).

All algorithms mentioned above for solving BOS store $f_1(n)$ and $f_2(n)$ for each node $n \in$*OPEN*. These algorithms typically order *OPEN* according to the lexicographic order of nodes. That is, the best node in *OPEN* is the node with the smallest f_1-value and in case of a tie then the smallest f_2-value. This allows BOA* to perform dominance checks in constant time (Hernandez et al. 2020). The implication is that solutions are found from the top left of the POF to its bottom right. In our setting we are only interested in finding one solution. This enables more flexibility in how we order nodes in *OPEN*. we describe several approaches next.

5 Choosing the Best Node from *OPEN*

Given the normalized bounds $\bar{b_1}$ and $\bar{b_2}$, our algorithm is required to search the rectangle whose extreme points are (0,0) and $(\bar{b_1}, \bar{b_2})$ (see Figure 1(c)). The POF points are diagonally crossing this rectangle in its top-right part. We are interested in a systematic approach that searches this rectangle in order to find one of these solutions as fast as possible. We now describe several strategies that allow to cover this rectangle (see Figure 1(d-h) for a visuaization of the different strategies). Such strategies will be defined using the notion of an *ordering function*. An ordering function O gets two single objective functions $\mathbf{f} = (f_1, f_2)$ and two nodes n and m. It returns the preferred node for expansion among n and m. It creates a total order between all nodes thereby determining the order of the nodes in *OPEN*. Let $O(F_1, F_2, n, m)$ be the following general ordering function where F_1 and F_2 are manipulations of f_1 and f_2.

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

We next describe a number of ordering functions which differ in how F_1 and F_2 are defined.

5.1 Lexicographic Orderings

The first ordering functions are lexicographic according to the first and second objectives, respectively. Namely,

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

The order of costs that are expanded using these variants is shown in Figure 1(d,e). O_{lex1} first scans the left vertical line (bottom up), then the second left etc. O_{lex2} first scans the bottom horizontal line (left to right) then the horizontal line above it etc. In the example of the figure, $\bar{b_1}$ is larger than $\bar{b_2}$. Since the POF points are crossing the top right part diagonally, O_{lex2} has a great advantage as it hits the bottom right POF point (point b in Figure 1(e)) after scanning less than half of the rectangle area (the area bellow it which is covered by the lines). By contrast, O_{lex1} scans the rectangle from left to right and needs to pass more than half of the rectangle before arriving at the first POT point at the top left point (point a). Therefore, when $\bar{b_1}$ is larger than $\bar{b_2}$ O_{lex2} is expected to outperform O_{lex1} and vice-versa.

5.2 Minimum and Maximum Orderings

Let $F_{\min}(n)$ and $F_{\max}(n)$ be two functions returning the maximal and minimal normalized f-value 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 Minimum and Maximum order (visualized in Figure 1(f,g)) as:

	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	1.0
$\bar{b}_2$	.95	.60	.51	.43	.21	.97	.74	.76	.62	.50	.98	.87	.84	.81	.75	1.0
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	147.0

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

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

$O_{\min}$ is combining both lexicographic orders while maximum acts more like a depth-first search.

5.3 Average Ordering

We next describe an ordering that is based on averaging the costs. Let $F_{\text{avg}}(n)$ be the average normalized cost of a node. Namely, $F_{\text{avg}}(n) := (\bar{f}_1(n) + \bar{f}_2(n))/2$. The average ordering O_{avg}, visualized in (Figure 1(h)) is defined as:

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

Note that in O_{avg} we break ties according to the smallest f-value of each node. We also tried other ordering functions, such as weighted average, $\bar{f}_1(n)^2 + \bar{f}_2(n)^2$ (circle shape), and $\bar{f}_1(n) \cdot \bar{f}_2(n)$ (hyperbola shape). These are not reported as we did not observe any benefit for these functions.

6 Experimental Results

We compared all our variants of BC-BOA* on the BAY road map (DIMACS 2006). A common practice is to use time and distance as objectives on this map (Hernandez et al. 2020). For the purpose of the experiments, we first found the POF for each problem instance. We then chose five *pivot* nodes from it 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 that is not an extreme solution. **(2) FBR**, which is similar to FTL for the farthest bottom-right solution. **(3) MD**, which is the POF solution that has the lowest difference between its c_1 and c_2 costs. **(4) TL**, which is the solution that is closest to the middle between MD and FTL. **(5) BR**, which is the solution that is closest to the middle between MD and FBR.

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

Table 1 presents the average number of node expansions over 135 problem instances for zones 2, 3, 4, and 5. Zones 2–4 are divided by the five pivots defined above. The first two rows present the averaged bounds used. Then, each row represents a different ordering function. We next use *Lex1* to denote BC-BOA* with the O_{lex1} ordering and so on. When comparing Lex1 and Lex2 we see that Lex1 was better in FTL and in TL while Lex2 was better in FBR and BR. For MD, Lex2 was better; this is probably some skewing of the exact map and the order of the parameters (time, distance). Based on this, *Selective LEX* is an intelligent variant which selects O_{lex2} if $\bar{b}_1 > \bar{b}_2$ and O_{lex1} otherwise. Thus, it exploits the benefits of both LEX variants. Our results confirm that Sel-LEX was the most robust variant: it was either the best variant or very close to it. Min also performed relatively well and was sometimes the best. The other variants performed poorly. Sel-LEX is therefore the winner variant.

Zone 5 is not divided by the pivots as this zone contains all POF and thus similar for all pivot. Zones 0 and 1 are not presented as all solvers need to expand the exact same number of nodes in these zones. In general, given two nodes $n_1 \prec n_2$ all our ordering methods expand n_1 before n_2. Therefore, when a single or no solution exists the same number of nodes are expanded.

Timing results are not reported because the constant time per node for all variants was very similar. Indeed, the Lex variants can exploit the constant-time dominance check of BOA*. While the other variants have a linear time dominance check, it is linear in the number of duplicates of a given state. Since we used the bound pruning, that number was very small and did not affect runtime.

When BC-BOA* continues to run, it will find *all* possible POF solutions within its bounds. The number of nodes expanded for this continuous variant is also presented (ALL POF) as well as the number of different solutions that this variant found. On average, there were 147 different POF solutions and 369K nodes are needed to find them. By contrast, the best variant for finding a single solution within a given bound reduced the number of nodes by a factor of 6 (Zone 2, TL) up to a factor of 500 (Zone 4, FBR).

7 Conclusions

We presented several variants for BC-BOA* and concluded that Selective Lex is the best choice. Future work can lift the requirement for a POF-solution thereby allowing variants of Potential Search (Stern, Puzis, and Felner 2011) to be activated here. The normalizing scheme can also be adopted in further algorithms. Finally, generalizing our work to $k > 2$ *multi-objective search* can be done.

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.

DIMACS. 2006. the 9th DIMACS Implementation Challenge: Shortest Path.

Ehrgott, M. 2005. *Multicriteria Optimization (2. ed.)*. Springer.

Fu, M.; Kuntz, A.; Salzman, O.; and Alterovitz, R. 2019. Toward Asymptotically-Optimal Inspection Planning Via Efficient Near-Optimal Graph Search. In *RSS*.

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.

Hernandez, C. U.; Yeohz, W.; Baier, J. A.; Zhang, H.; Suazoy, L.; and and, S. K. 2020. A Simple and Fast Bi-Objective Search Algorithm. In *ICAPS*.

Mandow, L.; and De La Cruz, J. L. P. 2005. A new approach to multiobjective 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 of 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.

Stern, R. T.; Puzis, R.; and Felner, A. 2011. Potential Search: A Bounded-Cost Search Algorithm. In Bacchus, F.; Domshlak, C.; Edelkamp, S.; and Helmert, M., eds., *Proceedings of the 21st International Conference on Automated Planning and Scheduling, ICAPS 2011, Freiburg, Germany June 11-16, 2011*. AAAI.

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