# Subset Approximation of Pareto Regions with Bi-objective A*

**Nicolás Rivera**[1], **Jorge A. Baier**[2,3], **Carlos Hernández**[4]

[1]Instituto de Ingeniería Matemática, Universidad de Valparaíso, Valparaíso, Chile
[2]Departamento de Ciencia de la Computación, Pontificia Universidad Católica de Chile, Santiago, Chile
[3]Instituto Milenio Fundamentos de los Datos, Santiago, Chile
[4]Facultad de Ingeniería y Tecnología, Universidad San Sebastián, Bellavista 7, 84205254, Santiago, Chile
n.a.rivera.aburto@gmail.com, jabaier@ing.puc.cl, carlos.hernandez@uss.cl,

**Abstract**

In bi-objective search, we are given a graph in which each directed arc is associated with a pair of non-negative weights, and the objective is to find the Pareto-optimal solution set. Unfortunately, in many practical settings, this set is too large, and therefore its computation is very time-consuming. In addition, even though bi-objective search algorithms generate the Pareto set incrementally, they do so exhaustively. This means that early during search the solution set covered is not diverse, being concentrated in a small region. To address this issue, we present a new approach to subset approximation of the solution set, that can be used as the basis for an anytime bi-objective search algorithm. Our approach transforms the given task into a target bi-objective search task using two real parameters. For each particular parameter setting, the solutions to the target task is a subset of the solution set of the original task. Depending on the parameters used, the solution set of the target task may be computed very quickly. This allows us to obtain, in challenging road map benchmarks, a rich variety of solutions in times that may be orders of magnitude smaller than the time needed to compute the solution set. We show that by running the algorithm with an appropriate sequence of parameters, we obtain a growing sequence of solutions that converges to the full solution set. We prove that our approach is correct and that Bi-Objective A* prunes at least as many nodes when run over the target task.

## 1 Introduction

In bi-objective search we are given a graph $G$ in which each arc, and thus each path, is associated with a pair of non-negative costs, which represent meaningful objective functions. For example, in transportation, one function could refer to the time required to traverse an edge while the other could refer to fuel consumption. To compare two paths, a *dominance* relation is used. Path $\pi_1$ dominates path $\pi_2$ if both components of the cost of $\pi_1$ are less than or equal to the respective components of the cost of $\pi_2$ and their costs are not equal. Given a start vertex and a goal vertex in $G$, the problem consists of finding a Pareto-optimal solution set which contains all paths from start to goal which are not dominated by another path from start to goal.

Bi-objective search is required for several real-world applications; notably in transportation and logistics when time and cost (e.g., fare) are minimized (e.g., Pallottino and Scutella 1998; Bronfman et al. 2015; Müller-Hannemann and Weihe 2006), or when time and risk are minimized for cycling (Ehrgott et al. 2012). Recently, it has also been used in AI problems like robot planning (Davoodi 2017) and multi-agent path finding (Ren, Rathinam, and Choset 2021).

An important hurdle to bi-objective search is the size of the solution set, which can be exponential on the size of the graph (Hansen 1980). As a consequence, bi-objective search algorithms may only compute a handful of solutions before running out of time. Worse even, because of the exhaustive nature of their search, such solutions may not represent the diversity of the solution set. To address this problem, approaches to approximating the solution set have been proposed. One line of work proposes algorithms that reduce high runtimes by computing a solution set with approximate solutions whose suboptimality is bounded (e.g., Warburton 1987; Perny and Spanjaard 2008; Goldin and Salzman 2021). Another less explored line of work computes *subset approximations* (e.g., Cohon 1978; Henig 1986), in which a subset of the solution set is computed. A limitation of the former approach is that even though approximate solutions may be faster to compute still a large number of solutions may have to be computed. The main limitation of the latter approach is that the maximum number of computable solutions is fixed and task-dependent. This does not allow returning more solutions if more search time is available.

In this paper we present a new approach to bi-objective subset approximation. Our approach transforms the original bi-objective search instance into another (target) bi-objective instance such that each solution to the target problem is a solution to the original problem. If a heuristic $\mathbf{h}$ is available for the original task our simple transformation is also applicable to the heuristic of the original problem, which allows solving the target task with existing bi-objective heuristic search algorithms. We prove that implicit to the target instance is a stricter dominance relation in the sense that a path $\pi$ in the target instance may dominate a path $\pi'$ in the target instance while the converse is not necessarily true. To generate the transformed instance we use two real parameters, $\alpha$ and $\beta$ in $(0, 1]$. By varying these parameters we obtain different subset approximations. When both parameters are equal to 1, we recover the original solution set.

While the target problem can be solved with any bi-

objective search algorithm, we test our approach by combining it with Bi-Objective A* (BOA*) (Hernndez et al. 2020), a recently proposed bi-objective heuristic search algorithm that was shown to outperform other algorithms as it scales better on large maps. We show that good synergy between the target problem(s) and BOA* exists; indeed, BOA* prunes more nodes as $\alpha$ and $\beta$ decrease their values. Empirically, we use standard roadmap benchmarks to show that, depending on the chosen parameters, BOA* can compute about 10–20% of the solutions in about one order of magnitude less time. In addition, we show that the solutions obtained are, on average, very diverse. As such, our approach shows promise for applications in which a few optimal but diverse solutions are required quickly by end users.

## 2 Background

### 2.1 Notation

A bi-objective *search graph* is a triple $(S, E, \mathbf{c})$, where $S$ is a finite set of *states*, $E \subseteq S \times S$ is a finite set of directed edges, and $\mathbf{c} : E \rightarrow \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$ is a non-negative *cost function*. For each $s \in S$ we denote by $\mathrm{Succ}(s) = \{t \in S \mid (s,t) \in E\}$ the successors of state $s$. A *path* $\pi$ 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 $i \in \{1, \ldots, n-1\}$.

Boldface lower case letters indicate column vectors in $\mathbb{R}^2$. The first and second component of $\mathbf{p}$ are denoted by $\mathbf{p}_1$ and $\mathbf{p}_2$, respectively. We consider standard addition and multiplication by scalar of vectors. We say that $\mathbf{p} \preceq \mathbf{q}$, iff $\mathbf{p}_1 \leq \mathbf{q}_1$ and $\mathbf{p}_2 \leq \mathbf{q}_2$; in addition, $\mathbf{p} \prec \mathbf{q}$ iff $\mathbf{p} \preceq \mathbf{q}$ and $\mathbf{p} \neq \mathbf{q}$. We say that $\mathbf{p}$ *dominates* $\mathbf{q}$ when $\mathbf{p} \prec \mathbf{q}$, and that $\mathbf{p}$ *weakly dominates* $\mathbf{q}$ when $\mathbf{p} \preceq \mathbf{q}$

Given a path $\pi = s_1, \ldots, s_n$, its cost is given by $\sum_{i=1}^{n-1} \mathbf{c}(s_i, s_{i+1})$ and denoted by $\mathbf{c}(\pi)$. Path $\pi$ dominates path $\pi'$ if and only if $\mathbf{c}(\pi) \prec \mathbf{c}(\pi')$.

A *bi-objective search instance* is as a tuple $(S, E, \mathbf{c}, s_0, s_g)$, where $(S, E, \mathbf{c})$ is a search graph, $s_0$ and $s_g$ are, respectively, the *start state* and the *goal state*. A *start-to-goal path* is a path from $s_0$ to $s_g$. The *Pareto-optimal solution set*, denoted by $\mathrm{sols}_P$, contains all start-to-goal paths that are not dominated by another one.

Bi-objective search algorithms exploit heuristic functions. The $\mathbf{h}$-value of a state is given by a function $\mathbf{h} : S \rightarrow \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$. For every $s \in S$, $\mathbf{h}(s)$ estimates the cost of a path from state $s$ to the goal state $s_g$. $\mathbf{h}$ is admissible if and only if for every state $s$, $\mathbf{h}(s) \preceq \mathbf{c}(\pi)$, where $\pi$ is any path from $s$ to $s_g$. Similarly, $\mathbf{h}$ is consistent if and only if, first, $\mathbf{h}(s_g) = \mathbf{0}$, and second, $\mathbf{h}(s) \preceq \mathbf{c}(s,t) + \mathbf{h}(t)$ for all $(s,t) \in E$.

### 2.2 Bi-Objective A*

In this section we introduce Bi-Objective A* (BOA*) (Hernndez et al. 2020). BOA* is a recent algorithm for solving bi-objective search instances, which was empirically shown to scale better to larger instances than other bi-objective search algorithms. One of the main advantages of BOA*, at least for our approach, is that it has the same algorithmic structure of A*, ensuring desirable theoretical properties (see for example Theorem 4 and Corollary 1 in Section Section 4.2 below).

Similar to A*, BOA* has a priority queue Open, which will allow us to decide which element shall be expanded in the implicit search tree. Open contain nodes, which shall not be confused with the states $S$. A state $s$ is an element of $S$ in the search graph (which may represent a city in a map), whereas a node $x$ in Open contains a state $s(x) \in S$ and a parent node $\mathrm{parent}(x)$. Therefore, a node $x$ represent the state $s(x)$, and the path that goes from $s_0$ to $s(x)$, which can be recovered by successive queries to parents nodes until we reach a node $y$ with $s(y) = s_0$ which has $\mathrm{parent}(y) = null$. We denote the path associated with node $x$ by $\pi_x$. Additionally, each node contains $\mathbf{g}$, $\mathbf{h}$ and $\mathbf{f}$-values as in the standard A* algorithm, whoever, these are bivariate functions. $\mathbf{g}(x)$ is the cost of the path $\pi_x$, i.e, $\mathbf{g}(x) = \mathbf{c}(\pi_x)$, $\mathbf{h}(s(x))$ is the heuristic value of state $s(x)$ (recall the heuristic is defined on the set of states $S$), and $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}(s(x))$.

BOA* starts the algorithm by initializing Open, with a node associated with the initial state $s_0$, which we sometimes refer to as the root node. The priority queue Open is sorted lexicographically with respect to the $\mathbf{f}$-values of the nodes, meaning that we prefer elements with smaller value in the first component, break ties using the second component. BOA* iterates by extracting the best candidates from Open, and expanding them, potentially including new elements to Open. The main difference between BOA* and A* with a consistent heuristic are: i) since in BOA* there are several non-dominated optimal solutions, states $s \in S$ may appear in several nodes in the open list with different non-dominated values of $\mathbf{f}$, ii) Since there are several non-dominated start-to-goal paths that we are interested on, when BOA* extracts $s_g$ from Open, it does not return (and end the execution), since there may be other solutions. Finally, iii) BOA* prunes the search tree to avoid consideration of paths that are dominated by others. Pruning of a node may occur right after its generation (i.e. before inserting it into Open), or right after its extraction from Open. To check for domination, BOA* exploits the fact that the heuristics used are consistent and that the open list is ordered lexicographically by $\mathbf{f}$-value. This implies that if a state is discovered by BOA* via several nodes, i.e. by different paths, the $\mathbf{f}_1$-values of such nodes are non-decreasing. This allows BOA* to implement dominance checks in constant time.

Now we go over the details of BOA*. Algorithm 1 shows its pseudo-code. In the initialization (Lines 1–8), the priority queue Open consists of a node associated with state $s_0$, with a $\mathbf{g}$-value equal to $\mathbf{0}$, and an $\mathbf{f}$-value given by its heuristic $\mathbf{h}(s_0)$. Such a node also has a parent pointer initialized to $null$ to indicate that the path towards $s_0$ is empty. Later, in the code parent pointers are used to define paths towards each node. In its main loop (Lines 9–24), like standard A*, BOA* extracts a node from Open (Line 10), and then expands it (Lines 17–24). Unlike A*, when a new solution is found, BOA* does not return but stores the newly found solution in sols (Lines 14–16).

Another aspect distinguishing BOA* from A* is its pruning. BOA*'s pruning is constant-time and it is key to its performance. To do so, for each state $s \in S$ it keeps a $g_2^{\min}$ value, corresponding to the minimum $\mathbf{g}_2$-value of a path to $s$ previously extracted from Open. Given that Open is sorted

**Algorithm 1:** Bi-Objective A* (BOA*)

**Input :** A search problem $(S, E, \mathbf{c}, s_0, s_g)$ and a consistent heuristic function $\mathbf{h}$
**Output:** A cost-unique Pareto-optimal solution set

```
1  sols ← ∅
2  for each s ∈ S do
3      g_2^min(s) ← ∞
4  x ← new node with s(x) = s_0
5  g(x) ← (0, 0)
6  parent(x) ← null
7  f(x) ← h(s_0)
8  Initialize Open and add x to it
9  while Open ≠ ∅ do
10     Remove a node x from Open with the
         lexicographically smallest f-value
11     if g_2(x) ≥ g_2^min(s(x)) or f_2(x) ≥ g_2^min(s_g) then
12         continue
13     g_2^min(s(x)) ← g_2(x)
14     if s(x) = s_g then
15         Add the path defined by x to sols
16         continue
17     for each t ∈ Succ(s(x)) do
18         y ← new node with s(y) = t
19         g(y) ← g(x) + c(s(x), t)
20         parent(y) ← x
21         f(y) ← g(y) + h(t)
22         if g_2(y) ≥ g_2^min(t) or f_2(y) ≥ g_2^min(s_g) then
23             continue
24         Add y to Open
25 return sols
```

lexicographically, each time a node $x$ (associated to state $s(x) \in S$) is extracted from Open, it is associated with a path whose $\mathbf{g}_1$-value is greater than or equal to the $\mathbf{g}_1$-value of any other path towards $s(x)$ previously extracted. This allows to focus pruning after extraction only on the second dimension, $\mathbf{g}_2$. Indeed, if $\mathbf{g}_2(x) \geq g_2^{\min}(s(x))$, it means node $x$ is either dominated by a previously found path or it has the same cost of a previously extracted path to $s(x)$. Thus, it is pruned in Line 11. The pruning condition of Line 11 is also made stronger by adding $\mathbf{f}_2(x) \geq g_2^{min}(s_g)$, which if violated would mean that the current path is dominated by a path towards the goal which has been previously extracted from Open. An analogous pruning condition is considered in Line 22, whose objective is to prune dominated nodes before they are added to Open.

## 3 Related Work: Subset Approximation

Now we focus on the problem of finding a subset of solutions. A well-known (Cohon 1978; Henig 1986) simple approach to obtain a reduced subset of solutions consists of mapping our given bi-objective search task $P = (S, E, \mathbf{c}, s_0, s_g)$ into a single-objective search problem $P_\alpha$ in which the (scalar) cost function is defined as $c = \alpha\mathbf{c}_1 + (1-\alpha)\mathbf{c}_2$, where $\alpha \in [0, 1]$. An interesting property is that a solution to $P_\alpha$ belongs to the Pareto-optimal solution set.

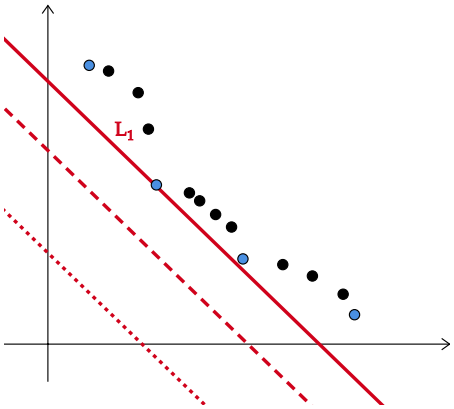


Figure 1: Black dots and blue dots represent a Pareto-optimal set of solutions of a bi-objective search instance. The red lines represent lines of the form $\alpha x + (1-\alpha)y = K$ for three different values of $K$. When $K = 0$ the line crosses the origin, and by increasing $K$ the line moves closer to the Pareto region until it hits it. Solving the problem $P_\alpha$ is equivalent to finding the minimum value $K$ that makes the line tangent to the Pareto region. Only blue points can be found by using this method.

**Theorem 1** (Henig (1986)). *Let $P = (S, E, \mathbf{c}, s_0, s_g)$ be a bi-objective search task. Let $\alpha \in [0, 1]$ and let $P_\alpha$ be the single-objective problem $(S, E, c, s_0, s_g)$, where $c = \alpha\mathbf{c}_1 + (1-\alpha)\mathbf{c}_2$ is a univariate cost function. If $\pi$ is a minimum-cost path for $P_\alpha$ then $\pi \in \text{sols}_P$.*

Perhaps the most interesting fact about this approach is its geometric interpretation. Indeed, finding a path with minimum cost given by $\alpha c_1 + (1-\alpha)c_2$ can be interpreted as finding the smallest value $K$ such that the line given by the equation $\alpha x + (1-\alpha)y = K$ contains a point in the Pareto set. In other words, finding a solution to $P_\alpha$ corresponds to finding a line parallel to $\alpha x + (1-\alpha)y = 1$ which is tangent to the Pareto set. See Figure 1 for an illustration.

To find a subset of solutions one may try various values of $\alpha$. But since the approach is limited to finding points which intersect with tangents to the solution set, it is limited to finding at most points in the convex hull of the solution set.

## 4 Solution Subsets via Bi-Objective Search

In this section we describe our approach to obtain a subset of the Pareto-optimal solution set. The approach can be used along with any bi-objective search algorithm. Our idea is to map the problem $P$ into another search problem $P_{\alpha,\beta}$ where $\alpha$ and $\beta$ are two parameters that control the precision of our approximation.

### 4.1 The problem $P_{\alpha,\beta}$

We shall assume that $\alpha, \beta \in (0, 1]$ with $\alpha + \beta > 1$. To build $P_{\alpha,\beta}$ we define the matrix $M_{\alpha,\beta}$ given by:

$$M_{\alpha,\beta} = \begin{pmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{pmatrix},$$

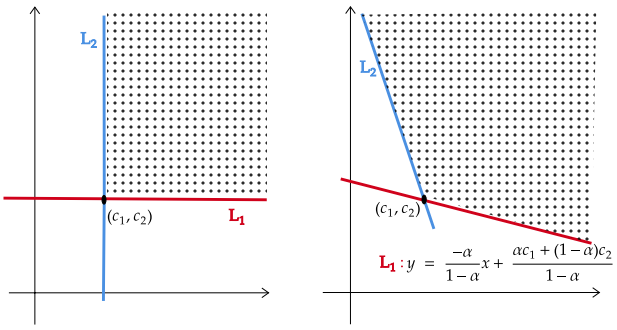


Figure 2: On the left we have the standard way to prune paths. Any path whose cost is inside the dotted region is pruned.

and define $P_{\alpha,\beta} = (S, E, \mathbf{c}_{\alpha,\beta}, s_0, s_g)$, where for each $e \in E$, we define $\mathbf{c}_{\alpha,\beta}(e)$ as the result of applying the matrix $M_{\alpha,\beta}$ to $\mathbf{c}(e)$ that is

$$M_{\alpha,\beta}\left(\mathbf{c}(e)\right) = (\alpha\mathbf{c}_1(e)+(1-\alpha)\mathbf{c}_2(e), (1-\beta)\mathbf{c}_1+\beta\mathbf{c}_2(e)).$$

Essentially, $P_{\alpha,\beta}$ is the same problem as $P$ but with its cost function multiplied by matrix $M_{\alpha,\beta}$. This new instance $P_{\alpha,\beta}$ has two important properties. The first property is that it defines a dominance relation $\preceq_{\alpha,\beta}$ in which $\mathbf{u}$ is dominated by $\mathbf{v}$ if and only if $\alpha\mathbf{u}_1+(1-\alpha)\mathbf{u}_2 \leq \alpha\mathbf{v}_1+(1-\alpha)\mathbf{v}_2$ and $(1-\beta)\mathbf{u}_1 + \beta\mathbf{u}_2 \leq (1-\beta)\mathbf{v}_1 + \beta\mathbf{v}_2$, and $\mathbf{u} \neq \mathbf{v}$, or much shorter, $M_{\alpha,\beta}\mathbf{u} \preceq M_{\alpha,\beta}\mathbf{v}$. (See Figure 2 for a graphical representation of the new dominance relation and for a formal geometrical interpretation see Proposition 1 below). Notice that when $\alpha = \beta = 1$ we recover the usual dominance relation. The second important property is that the solution set of $P_{\alpha,\beta}$ is contained in the solution set of $P$, as stated in Theorem 2 below. As a consequence, all solutions found when solving $P_{\alpha,\beta}$ can be reported as solution of $P$.

**Proposition 1.** *Let $L_1$ and $L_2$ be two lines defined by:*

$$L_1: \quad \alpha x + (1-\alpha)y = \alpha\mathbf{c}_1(\pi) + (1-\alpha)\mathbf{c}_2(\pi), \qquad (1)$$
$$L_2: \quad (1-\beta)x + \beta y = (1-\beta)\mathbf{c}_1(\pi) + \beta\mathbf{c}_2(\pi). \qquad (2)$$

*Let $\pi$ and $\pi'$ be two paths such that $\mathbf{c}(\pi) \neq \mathbf{c}(\pi')$. Then $\pi \prec_{\alpha,\beta} \pi'$ iff $\mathbf{c}(\pi')$ is in the intersection of the semi-plane above $L_1$ and the semi-plane above $L_2$.*

**Theorem 2.** *Let $P = (S, E, \mathbf{c}, s_0, s_g)$ be a bi-objective search instance and let $P_{\alpha,\beta}$ be defined as above with $\alpha, \beta \in (0, 1]$ and $\alpha + \beta > 1$. Then $\text{sols}_{P_{\alpha,\beta}} \subseteq \text{sols}_P$.*

To focus our presentation on the main ideas, we defer all proofs to Section 7.

## 4.2 Using $P_{\alpha,\beta}$ with BOA*

Theorem 2 states that solutions to $P_{\alpha,\beta}$ are solutions to $P$, and thus we can search solutions on the former problem to find solutions of the latter. In this search we will show that this idea works particularly well in combination with BOA* because the way BOA* navigates the search tree allow us not only to get results like Theorem 2 but also results ensuring that BOA* solving $P_{\alpha,\beta}$ will prune strictly more than BOA* solving $P$.

Before analyzing BOA* on the problem $P_{\alpha,\beta}$ we need to recall that BOA* is a heuristic-search algorithm, and so we need to find an heuristic. We start by assuming that we already have a heuristic function $\mathbf{h}$ for the original problem $P$. Then, we can obtain a heuristic for the new problem by applying $M_{\alpha,\beta}$ to the original heuristic $\mathbf{h}$. Henceforth, we denote by $\mathbf{h}_{\alpha,\beta}$ the result of applying $M_{\alpha,\beta}$ to $\mathbf{h}$, i.e. the bivariate function $(\alpha\mathbf{h}_1 + (1-\alpha)\mathbf{h}_2, \beta\mathbf{h}_1 + (1-\beta)\mathbf{h}_2)$. Similarly, we denote $\mathbf{g}_{\alpha,\beta}$, $\mathbf{f}_{\alpha,\beta}$, and $\mathbf{c}_{\alpha,\beta}$ the results of applying $M_{\alpha,\beta}$ to $\mathbf{g}, \mathbf{f}$ and $\mathbf{c}$, respectively. Note that by linearity of matrix multiplication $\mathbf{f}_{\alpha,\beta} = \mathbf{g}_{\alpha,\beta} + \mathbf{h}_{\alpha,\beta}$

**Theorem 3.** *Let $P$ be a search instance, and $\mathbf{h}$ be a heuristic function for $P$. Then (i) if $\mathbf{h}$ is admissible for $P$, then so is $\mathbf{h}_{\alpha,\beta}$ for $P_{\alpha,\beta}$, and (ii) if $\mathbf{h}$ is consistent for $P$, then so is $\mathbf{h}_{\alpha,\beta}$ for $P_{\alpha,\beta}$.*

As previously mentioned, another important property is that BOA* performs more pruning on $P_{\alpha,\beta}$ than on the original problem $P$. To understand this notion precisely recall that the problem $P_{\alpha,\beta}$ have the same underlying graph structure, independently of the value of $\alpha$ and $\beta$, and so they have the same search trees (with different edge-cost). In particular, all the problems $P_{\alpha,\beta}$ share the same set of paths from $s_0$, and so, since every node $x$ is uniquely characterized by the path $\pi_x$ leading to it from $s_0$, we conclude that node $x$ can also appear in the other problem (with different value of $\mathbf{f}$, $\mathbf{g}$, and $\mathbf{h}$) represented by the same path $\pi_x$. Therefore, we can refer to the same nodes for different problems $P_{\alpha,\beta}$ (using different $\alpha$ and $\beta$) by means of the path leading to them, and so we can talk about the same nodes in different problems.

To distinguish between executions of BOA* on $P$ or $P_{\alpha,\beta}$, we write BOA*$(P, \mathbf{h})$ and BOA*$(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$ to make clear what problem is solved and what is the input heuristic.

Additionally, we index all possible nodes in the search trees (before any prunes) with a unique natural number, which is used as tie-breaking mechanism in case two nodes have exactly the same $\mathbf{f}$ value in Open (preferring the node with less index for extraction). This is done so extraction of nodes is consistent in different problems $P_{\alpha,\beta}$. One way to implement this is to enumerate states in $S$, then we break $\mathbf{f}$-ties between nodes $x$ and $y$ by comparing $\pi_x$ and $\pi_y$, first by the number of states in the path, and if they have the same number of states we use the lexicographic order of the path given by the enumeration of the states in it.

Finally, in the following results we say that a node $z$ is **directly pruned** if such node satisfies the conditions of Line 22 or Line 11 in the respective query, and we say that a node $z$ is **pruned** if $z$ is directly pruned, or $\text{parent}(z)$ is pruned (if $\text{parent}(z) = null$, then $\text{parent}(z)$ is not pruned). Essentially, a vertex is pruned if it is directly pruned or a node in the backwards path to the root node is pruned.

**Theorem 4.** *Let $\alpha, \beta \in (0, 1]$ be such that $\alpha + \beta > 1$. Let $\mathbf{h}$ be a consistent heuristic. Then, if node $y$ is directly pruned in the execution of* BOA*$(P, \mathbf{h})$, *we have that $y$ is pruned in the execution of* BOA*$(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$,*

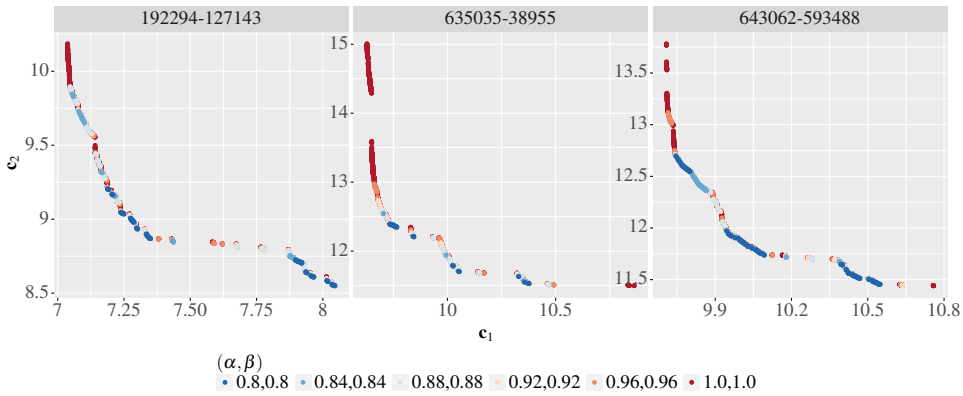


Figure 3: Pareto-optimal solutions of the three problems considered in our experiments. Reddish points are solutions that appear for small values of $\alpha$ and $\beta$, whereas bluish points are solutions that only appear for large values. Recall that for increasing values of $\alpha$ and $\beta$ solution sets grow. The title of each graph indicates the start and goal node. Values of $c_1$ and $c_2$ are in millions.

This theorem essentially tells us that after all the pruning performed in Line 22 and Line 11, the search tree of BOA*$(P, \mathbf{h})$ contains the search tree (after prunes) of BOA*$(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$ when $\mathbf{h}$ is consistent. This ensures that no more search is required in BOA*$(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$, and indeed, in the experimental section we will see that these new instances take much less time to be solved compared with the original problem, even when $\alpha$ and $\beta$ are very close to 1.

**Corollary 1.** *Let $\alpha_1, \alpha_2, \beta_1, \beta_2 \in (0, 1]$ be such that $1 \leq \alpha_1 + \beta_1$, and $\alpha_1 \leq \alpha_2$, and $\beta_1 \leq \beta_2$. Also, suppose that $\mathbf{h}$ is a consistent heuristic for problem $P$. Then, if node $y$ is directly pruned by* BOA*$(P_{\alpha_2,\beta_2}, \mathbf{h}_{\alpha_2,\beta_2})$*, we have that $y$ is pruned in the execution of* BOA*$(P_{\alpha_1,\beta_1}, \mathbf{h}_{\alpha_1,\beta_1})$.

## 5 Towards Anytime Bi-Objective Search

Our theoretical analysis implies that when using small values of $\alpha$ and $\beta$ more pruning is performed, and thus we shall expect faster executions. As a consequence, we propose a simple approach leading to an anytime bi-objective search algorithm, which is used later in Section 6. The main idea is to solve the target task for an initial pair of values, e.g. $\alpha = \beta = 0.8$, to then increase both parameters, and repeat until we reach $\alpha = \beta = 1$.

In general, different values of $\alpha$ and $\beta$ generate different solution sets, and the geometry of such sets depends not only on $\alpha$ and $\beta$ but also on the geometry of the original frontier (when $\alpha = \beta = 1$). Therefore, we are interested on measuring how many solutions of $\text{sols}_P$ are captured by $P_{\alpha,\beta}$, and whether or not these solutions are well-distributed. For this, we consider three different instances of the FL benchmark set (described in Section 6), whose frontiers are illustrated in Figure 3. The heatmap of Figure 4 shows, as expected, that for larger values of $\alpha$ and $\beta$ we cover more solutions, however, interestingly, increasing $\alpha$ has a greater impact on the number of solutions. This may be a consequence of the fact that BOA* finds solutions in increasing $\mathbf{c}_1$ order.

Regarding the distribution of the solutions, we observe it strongly depends on the structure of the solution costs. Figure 3 shows the distribution of the solutions for some values of $(\alpha, \beta)$. We observe that for the instance on the left, where the frontier is more balanced, solutions are produced in an even way, even for small values of $\alpha$ and $\beta$, whereas for the other two instances, the region of the solution set in which the first coordinate is small is not generated until the values of $\alpha$ and $\beta$ equal to 1.

This analysis illustrates that the way we vary $\alpha$ and $\beta$ has an important impact on solution diversity. Moreover, how diverse the solution set is given specific $\alpha$ and $\beta$ values is instance-specific. Below, in our experimental section, we use a general rule to increase $\alpha$ and $\beta$ that, we show, generates diverse solutions *on average*. We leave the problem of devising a sequence of $\alpha$-$\beta$ values for maximizing diversity at an instance level out of the scope of this paper.

## 6 Experimental Evaluation

Our experimental evaluation had the objective of evaluating the performance of BOA* run over $P_{\alpha,\beta}$ using different $(\alpha, \beta)$ values over a large number of instances in several standard road maps.

We evaluated our approach, implemented in C, on maps of the 9th DIMACS Implementation Challenge: Shortest Path[1]; specifically, 50 random instances for each of four USA road maps used by Machuca and Mandow (2012). We ran all experiments on a 3.80GHz Intel(R) Core(TM) i7-10700K CPU Linux machine with 64GB of RAM. The cost components represent travel distances ($c_1$) and times ($c_2$). The input heuristic $\mathbf{h}$ corresponds to the exact travel distances and times to the goal state, computed with Dijkstra's algorithm.

**Dividing the Pareto Frontier in Buckets** To report the diversity of solutions, imagine that we divide the upper-right quadrant of the plane (i.e. when both coordinates are positive) into five slices by drawing rays starting at the origin forming 18, 36, 54, and 72 degrees with the $x$-axis. We call each 18-degree slice a "bucket". By counting how many so-

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

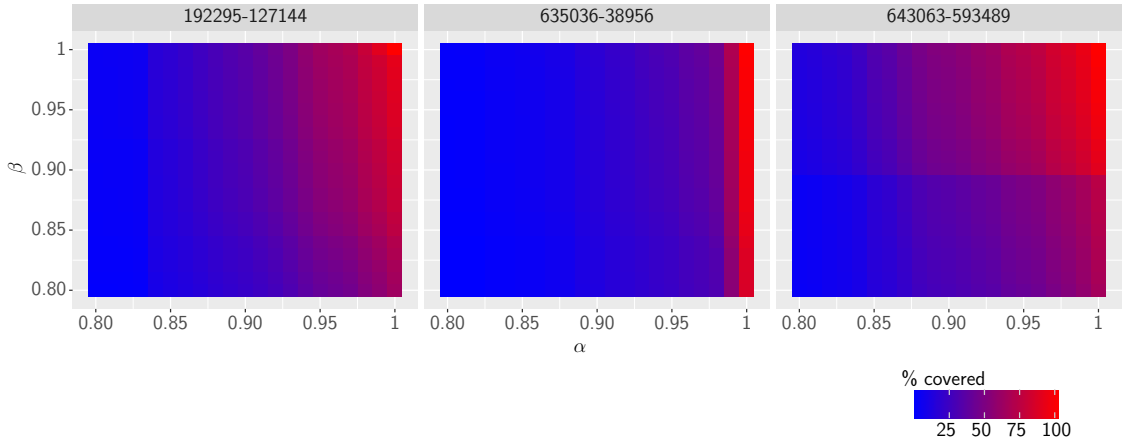


Figure 4: Heatmap for the three instances of the FL map. The color represents the solutions of $P$ covered by $P_{\alpha,\beta}$. In all instances we observe that increasing $\alpha$ has much more impact in the region we find, and indeed, choosing $\beta = 1$ while keeping a small value of $\alpha$ does not increase much the number of solutions. The title of each figure indicates the start and goal node.

lutions are in each bucket we obtain a measure of diversity of the solution set. However, to apply this idea it is necessary to adjust scales. Indeed each cost function may have a different scale and between different problems the solution sets may have different height and width[2]. Intuitively the objective is to "scale" the solution set such that its width and height are both equal to 1; as such, the extreme solutions are always $(0, 1)$ and $(1, 0)$. Now let $(x, y)$ be a scaled solution cost and $\theta$ be the angle between the origin and $(x, y)$. We count this solution in the bucket given by $\theta$.

Table 1 reports our results for 50 random instances for each road map. In each run we consider the same value for $\alpha$ and $\beta$. The table reports the total runtime in seconds required to compute the solution set for $P_{\alpha,\beta}$ for each $(\alpha, \beta)$ pair, and the percentage of solutions that appear in each bucket. In addition, the table reports the total number of solutions in each bucket. We have the following observations:

- We obtain solutions that are diverse on average. The maximum percentage difference between two buckets is 13%, which is observed in the BAY map with $\alpha = \beta = 0.84$
- A reasonable number of solutions can be obtained very quickly. For example, around 10% of solutions are obtained in about one order of magnitude less time than that required to find all solutions.
- When $(\alpha, \beta)$ values increase, the runtime increases and the number of solutions found in each bucket increases.
- The relation between computation time and percentage of solutions does not appear to be proportional as one would expect. For example in the FL map we compute approximately $8\%$ of the solutions in about $9.7/210.9 = 4.6\%$ of the time that is needed to compute $100\%$ of solutions. Likewise, to compute around $15\%$ of the solutions we require $16.8/210.9 = 8\%$ of the time required to compute $100\%$ of the solutions.

[2]Where one can imagine the width as the first component of the solution that has the largest first component, and the height can be defined analogously for the second component.

## 7 Proofs

In this section we provide a proof for each of our theoretical results. We begin with a simple lemma that is used in different proofs across this section.

**Lemma 1.** *Let $\alpha, \beta \in (0, 1]$, and $\mathbf{v}, \mathbf{u} \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$. If $\mathbf{v} \preceq \mathbf{u}$, then $M_{\alpha,\beta}\mathbf{v} \preceq M_{\alpha,\beta}\mathbf{u}$, and the same holds if we replace $\leq$ by $\prec$.*

*Proof.* We just prove the result for $\preceq$ since for the other case the same proof works.

Note that $\mathbf{u}_1 - \mathbf{v}_1 \geq 0$ and $\mathbf{u}_2 - \mathbf{v}_2 \geq 0$. Therefore $\alpha(\mathbf{u}_1 - \mathbf{v}_1) + (1 - \alpha)(\mathbf{u}_2 - \mathbf{v}_2) \geq 0$, since $\alpha > 0$. Hence $\alpha\mathbf{v}_1 + (1 - \alpha)\mathbf{v}_2 \leq \alpha\mathbf{u}_1 + (1 - \alpha)\mathbf{u}_2$. The same applies if we replace $\alpha$ by $1 - \beta$, concluding the result. □

Now we are ready to prove Theorem 1.

*Proof of Theorem 1.* Assume $\pi \notin \text{sols}_P$. Since $\pi$ is a start-to-goal path, there must be another path $\pi' \in \text{sols}_P$ such that $\mathbf{c}(\pi') \prec \mathbf{c}(\pi)$, so Lemma 1 yields

$$\alpha c_1(\pi') + (1 - \alpha)c_2(\pi') < \alpha c_1(\pi) + (1 - \alpha)c_2(\pi)$$

implying that $\pi$ is not a solution of $P_\alpha$, a contradiction. □

**Proof of Section 4.1.**

*Proof of Theorem 2.* For a contradiction, assume that there is a path $\pi \in \text{sols}_{P_{\alpha,\beta}}$ such that $\pi \notin \text{sols}_P$. Since $\pi$ is a start-to-goal path in $P$ but not a solution, there must be start-to-goal path $\pi'$ such that $\mathbf{c}(\pi') \prec \mathbf{c}(\pi)$. Then by Lemma 1, we have that $\mathbf{c}_{\alpha,\beta}(\pi') \prec \mathbf{c}_{\alpha,\beta}(\pi)$ which contradicts the fact that $\pi \in \text{sols}_{P_{\alpha,\beta}}$. □

| | | Bucket | | | | |
|---|---|---|---|---|---|---|
| $\alpha = \beta$ | $t(s)$ | 1 | 2 | 3 | 4 | 5 |
| **New York City (NY): 264,346 states, 730,100 edges** | | | | | | |
| 0.80 | 0.5 | 15% | 19% | 18% | 18% | 21% |
| 0.84 | 0.7 | 22% | 25% | 25% | 27% | 32% |
| 0.88 | 1.1 | 34% | 37% | 36% | 36% | 42% |
| 0.92 | 1.7 | 53% | 54% | 57% | 60% | 65% |
| 0.96 | 2.5 | 68% | 67% | 73% | 74% | 76% |
| 1 | 3.9 | 100% | 100% | 100% | 100% | 100% |
| # solutions: | | 1,924 | 2,744 | 3,370 | 2,001 | 1,931 |
| **San Francisco Bay (BAY): 321,270 states, 794,830 edges** | | | | | | |
| 0.80 | 0.5 | 4% | 7% | 11% | 13% | 17% |
| 0.84 | 0.8 | 8% | 12% | 17% | 16% | 21% |
| 0.88 | 1.4 | 16% | 20% | 26% | 22% | 26% |
| 0.92 | 2.4 | 28% | 33% | 38% | 32% | 39% |
| 0.96 | 4.0 | 48% | 54% | 59% | 49% | 56% |
| 1 | 8.5 | 100% | 100% | 100% | 100% | 100% |
| # solutions: | | 3,509 | 2,989 | 3,343 | 2,230 | 2,245 |
| **Colorado (COL): 435,666 states, 1,042,400 edges** | | | | | | |
| 0.80 | 1.3 | 5% | 5% | 4% | 6% | 15% |
| 0.84 | 2.1 | 9% | 10% | 6% | 8% | 18% |
| 0.88 | 3.7 | 15% | 15% | 9% | 13% | 24% |
| 0.92 | 7.2 | 27% | 26% | 20% | 23% | 31% |
| 0.96 | 15.6 | 45% | 44% | 36% | 41% | 45% |
| 1 | 46.5 | 100% | 100% | 100% | 100% | 100% |
| # solutions: | | 8,272 | 8,140 | 6,946 | 4,937 | 6,431 |
| **Florida (FL): 1,070,376 states, 2,712,798 edges.** | | | | | | |
| 0.80 | 9.7 | 7% | 6% | 9% | 8% | 11% |
| 0.84 | 16.8 | 13% | 11% | 15% | 18% | 20% |
| 0.88 | 32.2 | 22% | 20% | 26% | 32% | 31% |
| 0.92 | 50.8 | 35% | 36% | 40% | 46% | 45% |
| 0.96 | 85.4 | 60% | 61% | 66% | 71% | 68% |
| 1 | 210.9 | 100% | 100% | 100% | 100% | 100% |
| # solutions: | | 13,019 | 16,817 | 17,143 | 8,689 | 11,009 |

Table 1: Results on 50 random instances for different road maps. In all these experiments we set $\alpha = \beta$. The table shows the $\alpha$ (and $\beta$), and the total runtime to solve all the instances, and the percentage of solutions in each bucket.

**Proofs of Section 4.2** We omit the proof of Theorem 3 as it is a straightforward consequence of Lemma 1, and the definition of admissible and consistent. For the proof of Theorem 4 we require the following lemmas.

**Lemma 2.** *[(Hernndez et al. 2020, Lemma 2)] The sequences of extracted nodes and of expanded nodes have monotonically non-decreasing $\mathbf{f}_1$-values.*

**Lemma 3.** *Consider* $\mathrm{BOA}^*(P, \mathbf{h})$. *Let $y$ be a node such that it satisfies the condition of Line 11 or Line 22 during the execution of* $\mathrm{BOA}^*(P, \mathbf{h})$*, i.e. $y$ was directly pruned after removing it from the open list, or after expanding its parent (preventing $y$ from being inserted in* Open*). Then there exists a node $z$ such that $s(y) = s(z)$ or $s(z) = s_g$, and such that $\mathbf{f}(z) \preceq \mathbf{f}(y)$*

*Proof.* We analyze the case that $y$ is directly pruned in Line 22, as the case of Line 11 is essentially the same. Suppose that $y$ is directly pruned in Line 22. If $y$ is directly pruned because the first clause of the **if** statement, then let $t = s(y)$ be the state associated to node $y$, and so we have that some node $z$ with $s(z) = t$ was expanded in Line 13 before $y$ giving that current value of $g_2^{\min}(t)$, i.e. $\mathbf{g}_2(y) \geq \mathbf{g}_2(z)$, and thus, $\mathbf{f}_2(y) \geq \mathbf{f}_2(z)$ since $s(y) = s(z) = t$ (as both nodes share the same heuristic value $\mathbf{h}(y)$). Since node $z$ was extracted before $y$, by Lemma 2, $\mathbf{f}_1(z) \leq \mathbf{f}_2(y)$, concluding that $\mathbf{f}(z) \preceq \mathbf{f}(y)$. If $y$ is directly pruned because of the second clause of the **if** statement, the argument is the same but in this case $s(z) = s_g$ and so $\mathbf{h}(s(z)) = \mathbf{0}$. □

*Proof of Theorem 4.* We proceed by contradiction: suppose it exists a node $y$ which is directly pruned in $\mathrm{BOA}^*(P, \mathbf{h})$ in Line 11 or Line 22, but it is not pruned in $\mathrm{BOA}^*(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$, i.e. $y$ is included in the open list, and then extracted, and expanded in $\mathrm{BOA}^*(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$.

Since $y$ is directly pruned in $\mathrm{BOA}^*(P, \mathbf{h})$, by Lemma 3, there exists a node $z$ such that $s(y) = s(z)$ or $s(z) = s_g$, and such that $\mathbf{f}(z) \leq \mathbf{f}(y)$. By Lemma 1, $\mathbf{f}_{\alpha,\beta}(z) \prec \mathbf{f}_{\alpha,\beta}(y)$ or $\mathbf{f}_{\alpha,\beta}(z) = \mathbf{f}_{\alpha,\beta}(y)$, and thus it has smaller (or equal) first-coordinate in $\mathbf{f}_{\alpha,\beta}$. In case $z$ and $y$ have the same $\mathbf{f}$-value, by the tie-breaking mechanism implemented $z$ will be extracted before from Open than $y$, so we can assume that $\mathbf{f}_{\alpha,\beta}(z) \prec \mathbf{f}_{\alpha,\beta}(y)$. From now on, we shall focus on the behavior of node $z$ in the execution of $\mathrm{BOA}^*(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$. In the execution, node $z$ either i) enters to the open list of $\mathrm{BOA}^*(P_{\alpha,\beta}, \mathbf{h}_{\alpha,\beta})$ and it is successfully removed (and expanded), or ii) not. If it does (**case i**), then node $z$ is extracted and expanded before node $y$ due to Lemma 2, and hence $y$ will be directly pruned in Line 11 since we would have observed that $z$ have smaller second-component value in $\mathbf{g}_{\alpha,\beta}(z)$ (assuming $y$ is not pruned before), hence $y$ is pruned. If $z$ is pruned in either Line 11 or Line 22 (**case ii**), then it was pruned due to another node $x$ that was extracted from the open (i.e. BOA* read Line 13 with this node) and moreover $\mathbf{f}_{\alpha,\beta}(x) \preceq \mathbf{f}_{\alpha,\beta}(z) \prec \mathbf{f}_{\alpha,\beta}(y)$. Hence, $x$ is extracted from Open before $y$, and thus $y$ is pruned. □

*Proof of Corollary 1.* Let $\alpha$ and $\beta$ be defined as

$$\alpha = \frac{\alpha_1 + \beta_2 - 1}{\alpha_2 + \beta_2 - 1} \quad \text{and} \quad \beta = \frac{\beta_1 + \alpha_2 - 1}{\alpha_2 + \beta_2 - 1},$$

and note that $\alpha, \beta \in (0, 1]$ and $\alpha + \beta > 1$, due to the constrains on $\alpha_1, \alpha_2, \beta_1$ and $\beta_2$. Then, a straightforward computation shows that $M_{\alpha,\beta} M_{\alpha_2,\beta_2} = M_{\alpha_1,\beta_1}$.

Now we can assume that $P_{\alpha_2,\beta_2}$ was the original problem, and that we are applying the transformation $M_{\alpha,\beta}$ to it, obtaining $P_{\alpha_1,\beta_1}$. The result follows then from Theorem 4. □

## 8 Conclusions and Future Work

We presented a new approach for generating subset approximations of Pareto-optimal solution sets. Our approach transforms the given task into another bi-objective task which solution set is a subset of the Pareto-optimal solution set. This allows us to obtain a diverse solution set containing around 10% of solutions in one order of magnitude less time than what is needed to compute the whole Pareto-optimal set.

There are many future directions for research. Search solutions on average are diverse but we know that an instance level a more fine-grained approach to set $\alpha$ and $\beta$ is needed. Our approach also seems compatible with parallelization, which could yield very efficient anytime algorithms.

## Acknowledgements

Nicols Rivera was supported by ANID FONDECYT grant number 3210805. Jorge Baier and Carlos Hernndez are grateful to the Centro Nacional de Inteligencia Artificial CENIA, FB210017, BASAL, ANID.

## 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.

Cohon, J. L. 1978. Multiobjective Programming and Planning, volume 140 of. *Mathematics in Science and Engineering*.

Davoodi, M. 2017. Bi-objective path planning using deterministic algorithms. *Robotics and Autonomous Systems*, 93: 105–115.

Ehrgott, M.; Wang, J. Y.; Raith, A.; and Van Houtte, C. 2012. A bi-objective cyclist route choice model. *Transportation research part A: policy and practice*, 46(4): 652–663.

Goldin, B.; and Salzman, O. 2021. Approximate Bi-Criteria Search by Efficient Representation of Subsets of the Pareto-Optimal Frontier. In Biundo, S.; Do, M.; Goldman, R.; Katz, M.; Yang, Q.; and Zhuo, H. H., eds., *Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS)*, 149–158. AAAI Press.

Hansen, P. 1980. Bicriterion path problems. In *Multiple criteria decision making theory and application*, 109–127. Springer.

Henig, M. I. 1986. The shortest path problem with two objective functions. *European Journal of Operational Research*, 25(2): 281–291.

Hernndez, C.; Yeoh, W.; Baier, J.; Zhang, H.; Suazo, L.; and Koenig, S. 2020. A simple and fast bi-objective search algorithm. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, 143–151.

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

Müller-Hannemann, M.; and Weihe, K. 2006. On the cardinality of the Pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1): 269–286.

Pallottino, S.; and Scutella, M. G. 1998. Shortest path algorithms in transportation models: classical and innovative aspects. In *Equilibrium and advanced transportation modelling*, 245–281. Springer.

Perny, P.; and Spanjaard, O. 2008. Near Admissible Algorithms for Multiobjective Search. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, volume 178, 490–494.

Ren, Z.; Rathinam, S.; and Choset, H. 2021. Multi-objective Conflict-based Search for Multi-agent Path Finding. *CoRR*, abs/2101.03805.

Warburton, A. 1987. Approximation of Pareto optima in multiple-objective, shortest-path problems. *Operations Research*, 35(1): 70–79.