A-A*pex: Efficient Anytime Approximate Multi-Objective Search

Han Zhang[1], **Oren Salzman**[2], **Ariel Felner**[3], **Carlos Hernández Ulloa**[4,5,6], **Sven Koenig**[1]
[1] University of Southern California, USA
[2] Technion—Israel Institute of Technology, Israel
[3] Ben-Gurion University, Israel
[4] Universidad San Sebastián, Chile
[5] Centro Nacional de Inteligencia Artificial CENIA, Chile
[6] Centro Ciencia & Vida, Chile
zhan645@usc.edu, osalzman@cs.technion.ac.il, felner@bgu.ac.il, carlos.hernandez@uss.cl, skoenig@usc.edu

Abstract

In the multi-objective search problem, a typical task is to compute the Pareto frontier, i.e., the set of all undominated solutions. However, computing the entire Pareto frontier can be very time-consuming, and in practice, we often have limited deliberation time. Therefore, this paper focuses on solving the multi-objective search problem with anytime algorithms, which compute an initial approximate frontier quickly and then work to find more solutions until eventually finding the entire Pareto frontier. Existing work has investigated such anytime algorithms for problem instances with only two objectives. In this paper, we propose Anytime A*pex (A-A*pex), which works with any number of objectives. In each iteration of A-A*pex, it runs A*pex, a state-of-the-art approximate multi-objective search algorithm, to compute more solutions. From one iteration to the next, A-A*pex can either reuse its previous search effort or restart from scratch. Our experimental results show that an A-A*pex variant that mixes reusing its search effort and restarting from scratch yields the best runtime performance. We also show that A-A*pex often computes solutions that collectively approximate the Pareto frontier much better than the solutions found by state-of-the-art multi-objective search algorithms for short deliberation times.

1 Introduction

In multi-objective search, we are given a graph, a start state, and a goal state. The *cost* of each edge in the graph is a vector. Each component of the vector corresponds to a cost metric to minimize, such as travel time, travel distance, or risk. Multi-objective search is important for many real-world applications, including route planning for trucks, robots, and power lines (Bachmann et al. 2018) as well as inspecting regions of interest with robots (Fu et al. 2019; Fu, Salzman, and Alterovitz 2021). For example, transporting hazardous material requires one to trade-off between multiple costs for each street, such as its length and the number of residents that would be exposed to the hazardous material in case of a traffic accident (Bronfman et al. 2015).

In multi-objective search, the cost of a path is the component-wise sum of its edge costs. A path π *dominates* a path π' iff π is not worse than π' on any cost metric and is better than π' on at least one cost metric. A *solution* is a path from the start state to the goal state. A typical task of multi-objective search is to find the *Pareto frontier*, that is, all undominated solutions. Unfortunately, the size of the Pareto frontier can be exponential in the size of the graph being searched (Ehrgott 2005; Breugem, Dollevoet, and van den Heuvel 2017), which often makes computing the entire Pareto frontier time-consuming. Researchers have therefore proposed to find an *approximate frontier* instead (Tsaggouris and Zaroliagis 2009; Warburton 1987; Goldin and Salzman 2021; Zhang et al. 2022a; Perny and Spanjaard 2008), that is, a set of solutions such that any solution in the Pareto frontier is ε-dominated by some solution in the approximate frontier, where ε is a user-provided approximation factor. A path π *ε-dominates* a path π' for some $\varepsilon \geq 0$ if each cost component of $c(\pi)$ is no larger than $(1+\varepsilon)$ times the corresponding cost component of $c(\pi')$. The sizes of approximate frontiers are typically much smaller than those of the Pareto frontier (even for small approximation factors) and hence can be computed efficiently.

However, it remains unclear how to specify an approximation factor with which a search algorithm makes the best use of limited deliberation time. Therefore, in this paper, we investigate anytime approximate multi-objective search algorithms, which compute an initial approximate frontier quickly and then work to find better approximate frontiers until eventually finding the entire Pareto frontier. Existing works on anytime multi-objective search algorithms focus on problem instances with only two objectives. A-BOA*-ε (Zhang et al. 2022b) builds upon BOA*-ε, an approximate bi-objective search algorithm. It iteratively invokes BOA*-ε with decreasing approximation factors to compute new solutions. Additionally, each run of BOA*-ε reuses previous search effort by initializing BOA*-ε with nodes that were pruned in the previous run.

In this paper, we propose *Anytime A*pex* (A-A*pex), which builds upon A*pex and works with any number of objectives. A*pex is a state-of-the-art approximate multi-objective search algorithm that speeds up the search by merging similar search nodes. Zhang et al. (2022a) showed that A*pex outperforms PP-A* (Goldin and Salzman 2021), another approximate bi-objective search algorithm that has been shown to outperform BOA*-ε by up to an order of magnitude with respect to runtime . Different from PP-A* and

BOA*-ε, A*pex works with any number of objectives.

From one iteration to the next, A-A*pex can either reuse its previous search effort or restart the search from scratch. We propose a technique for reusing previous search effort by resuming the search from paths that were pruned in the previous iteration. Additionally, we propose a hybrid variant of A-A*pex which first restarts the search from scratch for each iteration and then starts to reuse its search effort in later iterations. Existing work on anytime single-objective search has investigated reusing search effort (Likhachev, Gordon, and Thrun 2003) or restarting from scratch (Richter, Thayer, and Ruml 2010). In this paper, we show how to reuse the search effort of A*pex despite its unique merge operations.

In our experimental results, we evaluate different variants of A-A*pex and show that reusing search effort in later iterations significantly reduces the runtime of A-A*pex. We also show that A-A*pex often computes solutions that collectively approximate the Pareto frontier much better than the solutions found by state-of-the-art multi-objective search algorithms for short deliberation times.

2 Terminology and Problem Definition

We use **boldface** font to denote vectors and v_i to denote the i-th component of a vector $\mathbf{v}$. The addition of two vectors $\mathbf{v}$ and $\mathbf{v}'$ of the same length N is defined as $\mathbf{v} + \mathbf{v}' = [v_1 + v'_1, v_2 + v'_2 \dots v_N + v'_N]$. We say that $\mathbf{v}$ *weakly dominates* $\mathbf{v}'$, denoted as $\mathbf{v} \preceq \mathbf{v}'$, iff $v_i \leq v'_i$ for all $i = 1, 2 \dots N$. We say that $\mathbf{v}$ *dominates* $\mathbf{v}'$, denoted as $\mathbf{v} \prec \mathbf{v}'$, iff $\mathbf{v} \preceq \mathbf{v}'$ and there exists an $i \in \{1, 2 \dots N\}$ with $v_i < v'_i$. For an *approximation factor* $\varepsilon \geq 0$, we say that $\mathbf{v}$ *ε-dominates* $\mathbf{v}'$, denoted as $\mathbf{v} \preceq_\varepsilon \mathbf{v}'$, iff $v_i \leq (1+\varepsilon)v'_i$ for all $i = 1, 2 \dots N$. The *truncated vector* of a vector $\mathbf{v}$, denoted as $Tr(\mathbf{v})$, is $\mathbf{v}$ with its first component deleted, i.e., $[v_2, v_3 \dots v_N]$.

A *(multi-objective search) graph* is a tuple $G = \langle S, E, \mathbf{c} \rangle$, where S is a finite set of *states* and $E \subseteq S \times S$ is a finite set of *edges*. $\text{outEdges}(s) = \{\langle s, s' \rangle : \langle s, s' \rangle \in E\}$ denotes the out-edges of a state s. Cost function $\mathbf{c} : E \to \mathbb{R}^N_{>0}$ maps an edge to its cost, which is a vector with N non-negative components. A *(multi-objective search) problem instance* is a tuple $P = \langle G, s_{\text{start}}, s_{\text{goal}} \rangle$, where G is a graph, $s_{\text{start}} \in S$ is the start state, and $s_{\text{goal}} \in S$ is the goal state.

A *path* from state s_1 to state s_l is a sequence of states $\pi = [s_1, s_2 \dots s_l]$ with $\langle s_i, s_{i+1} \rangle \in E$ for all $i = 1, 2 \dots l-1$. $s_1 = s_{\text{start}}$ unless mentioned otherwise. $\mathbf{c}(\pi) = \sum_{i=1}^{l-1} \mathbf{c}(\langle s_i, s_{i+1} \rangle)$ denotes the cost of path π. Path π can be *extended* with an edge $\langle s_l, s_{l+1} \rangle$ to obtain a new path $[s_1, s_2 \dots s_l, s_{l+1}]$. Path π *dominates* (resp. *weakly dominates* and *ε-dominates*) another path π' iff $\mathbf{c}(\pi) \prec \mathbf{c}(\pi')$ (resp. $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi')$ and $\mathbf{c}(\pi) \preceq_\varepsilon \mathbf{c}(\pi')$). A *solution* is a path from s_{start} to s_{goal}. A *Pareto-optimal* solution is a solution that is not dominated by any other solution.

A *(cost-unique) Pareto frontier* is a maximal subset of all Pareto-optimal solutions such that any two solutions in the subset do not have the same cost. An *ε-approximate frontier* is a set of solutions Π_ε such that, for any Pareto-optimal solution π', there exists a solution $\pi \in \Pi_\varepsilon$ with $\pi \preceq_\varepsilon \pi'$. The Pareto frontier is an ε-approximate frontier for any ε-value but not necessarily vice versa.

We define the *dominance factor* of a solution π over another solution π' as

$$\text{DF}(\pi, \pi') = \max\left(\max_{i=1,2\dots N}\left\{\frac{c_i(\pi)}{c_i(\pi')} - 1\right\}, 0\right),$$

which measures how "good" π approximates π'. $\text{DF}(\pi, \pi')$ is the smallest ε-value that satisfies $\pi \preceq_\varepsilon \pi'$. For a set of solutions Π, we define the *approximation error* of Π over a solution π' as

$$e(\Pi, \pi') = \min_{\pi \in \Pi} \text{DF}(\pi, \pi').$$

Roughly speaking, we find a path π in Π that approximates π' the best and compute the dominance factor. We have $e(\Pi, \pi') = 0$ iff $\exists \pi \in \Pi, \pi \preceq \pi'$. Let Π^* denote the Pareto frontier. We define the *approximation error* of a set of solutions Π as

$$e(\Pi) = \max_{\pi \in \Pi^*} e(\Pi, \pi). \tag{1}$$

$e(\Pi)$ is the smallest ε-value for which Π is an ε-approximate frontier.

We are interested in finding a set of solutions with a small approximation error within a limited deliberation time. More specifically, we focus on the *anytime behavior* of a search algorithm, i.e., its ability to quickly reduce the approximation error over time and eventually find the Pareto frontier.

A *heuristic (function)* $\mathbf{h} : S \to \mathbb{R}^N_{\geq 0}$ estimates the cost from a given state to the goal state. We assume that heuristic $\mathbf{h}$ is *consistent*, that is, $\mathbf{h}(s_{\text{goal}}) = \mathbf{0}$ and $\mathbf{h}(s) \preceq \mathbf{c}(\langle s, s' \rangle) + \mathbf{h}(s')$ for all $\langle s, s' \rangle \in E$.

3 Algorithmic Background

In this section, we review the existing multi-objective search algorithms BOA* (Hernández et al. 2023), A-BOA*-ε (Zhang et al. 2022b), and A*pex (Zhang et al. 2022a). For additional background, we refer the reader to a recent survey (Salzman et al. 2023). All of the aforementioned algorithms conform to the same *best-first multi-objective search* framework: A *(search) node* n contains a state $s(n)$ and a $\mathbf{g}$-value $\mathbf{g}(n)$ and corresponds to a path from s_{start} to $s(n)$, called the *path of* n. The $\mathbf{f}$-value of n is defined as $\mathbf{f}(n) = \mathbf{g}(n) + \mathbf{h}(s(n))$. The search algorithm maintains a priority queue $Open$ for generated but not expanded nodes and a set of solutions $Sols$. $Open$ is initialized with a node containing state s_{start} and $\mathbf{g}$-value $\mathbf{0}$.

In each iteration, the search algorithm *extracts* a node n from $Open$ with the *lexicographically smallest* $\mathbf{f}$-value. It then performs *dominance checks* to determine if it can prune the node. If not, it then *expands* n: If $s(n) = s_{\text{goal}}$, then the search algorithm adds the path of n, which is a solution, to $Sols$. Otherwise, it generates a new child node n' for each edge in $\text{outEdges}(s(n))$. It then performs dominance checks to determine if it can prune n' and, if not, adds n' to $Open$. When $Open$ becomes empty, the search algorithm terminates and returns $Sols$.

Best-first multi-objective search algorithms differ mainly in which information is contained in the nodes and how they perform dominance checks.

3.1 BOA* and A-BOA*-ε

BOA* (Hernández et al. 2023) is a bi-objective search algorithm that computes a Pareto frontier. In BOA*, each node n corresponds to a path π from s_{start} to $s(n)$ whose cost is $\mathbf{g}(n)$, and the child node of n for an out-edge e of n corresponds to the path that extends π by e. In its dominance checks, BOA* prunes a node n iff there exists (Condition 1) an expanded node n' containing the same state as n (i.e., the paths of n and n' have the same last state) with $\mathbf{g}(n') \preceq \mathbf{g}(n)$ or (Condition 2) an expanded node n' containing state s_{goal} (i.e., n' corresponds to a solution) with $\mathbf{g}(n') \preceq \mathbf{f}(n)$. BOA* uses dimensionality reduction (Pulido, Mandow, and Pérez-de-la Cruz 2015) to speed up its dominance checks: By exploiting the fact that the nodes extracted from $Open$ have lexicographically non-decreasing $\mathbf{f}$-values, Conditions 1 and 2 can be checked in constant time by checking if $g_2^{\min}(s(n)) \leq g_2(n)$ and $g_2^{\min}(s_{\text{goal}}) \leq f_2(n)$, respectively, where $g_2^{\min}(s)$ is the minimum g_2-value of all expanded nodes containing state s (Hernández et al. 2023).

BOA*-ε (Goldin and Salzman 2021) is a variant of BOA* that computes an ε-approximate frontier. BOA*-ε relaxes Condition 2 of the dominance checks in BOA* and prunes a node n if $g_2^{\min}(s_{\text{goal}}) \leq (1+\varepsilon) \cdot f_2(n)$.

A-BOA*-ε (Zhang et al. 2022b) is an anytime approximate bi-objective search algorithm which calls a variant of BOA*-ε to reduce the approximation error over time and eventually find the entire Pareto frontier. A-BOA*-ε stores those pruned nodes that might still lead to Pareto-optimal solutions and resumes its search from these nodes (by initializing $Open$ of BOA*-ε with them) in each iteration.

3.2 A*pex

Like BOA*-ε, A*pex computes an ε-approximate frontier, but, unlike BOA*-ε, it works with any number of objectives. In A*pex, a node is a so-called *apex-path pair* $\mathcal{AP} = \langle \mathbf{A}, \pi \rangle$ that consists of a cost vector $\mathbf{A}$, called the *apex*, and a path π, called the *representative path*. We define the $\mathbf{g}$-value of $\mathcal{AP}$ as $\mathbf{g}(\mathcal{AP}) = \mathbf{A}$ and $s(\mathcal{AP})$ as the last state of the representative path π. The $\mathbf{f}$-value of $\mathcal{AP}$ is $\mathbf{f}(\mathcal{AP}) = \mathbf{g}(\mathcal{AP}) + \mathbf{h}(s(\mathcal{AP}))$. Conceptually, an apex-path pair corresponds to a set of paths with the same last state, and its apex is the component-wise minimum of the costs of these paths. We define $\mathcal{AP}$ as *ε-bounded* iff $\mathbf{c}(\pi) + \mathbf{h}(s(\mathcal{AP})) \preceq_\varepsilon \mathbf{f}(\mathcal{AP})$.

Algorithm 1 shows the pseudo-code of A*pex. We reformulate the pseudo-code by Zhang et al. (2022a) and encapsulate part of A*pex in the findApproxPF function, which we will use to describe A-A*pex later.

A*pex performs dominance checks after generating an apex-path pair (Lines 19-20) and extracting an apex-path pair from $Open$ (Lines 10-11) using the isDominated function that prunes an apex-path pair $\mathcal{AP}$ iff there exists (Condition 1) an expanded apex-path pair $\mathcal{AP}'$ containing state $s(\mathcal{AP})$ with $\mathbf{g}(\mathcal{AP}') \preceq \mathbf{g}(\mathcal{AP})$ or (Condition 2) a solution π in $Sols$ that satisfies $\mathbf{c}(\pi) \preceq_\varepsilon \mathbf{f}(\mathcal{AP})$. A*pex uses dimensionality reduction (Pulido, Mandow, and Pérez-de-la Cruz 2015) to speed up the dominance checks by not checking the g_1-values for Condition 1. Instead of maintaining the $\mathbf{g}$-values of all expanded apex-path pairs, A*pex maintains only the often significantly smaller set of undominated truncated $\mathbf{g}$-values $G_{\text{cl}}^T(s)$ for each state s.

Algorithm 1: A*pex

```
   Input : P = ⟨G, s_start, s_goal⟩, ε, h
1  Open ← {⟨0, [s_start]⟩}
2  Sols ← ∅
3  findApproxPF(ε)
4  return Sols
5  Function findApproxPF(ε):
6      foreach s ∈ S do
7          G_cl^T(s) ← ∅
8      while Open ≠ ∅ do
9          AP = ⟨A, π⟩ ← Open.extract()
10         if isDominated(AP) then
11             continue
12         G_cl^T(s(AP)).add(Tr(g(AP))
13         if s(AP) = s_goal then
14             remove solutions weakly dominated by π
                 from Sols
15             add π to Sols
16             continue
17         for e ∈ outEdges(s(AP)) do
18             AP' ← ⟨A + c(e), extend(π, e)⟩
19             if isDominated(AP') then
20                 continue
21             addToOpen(AP')
22 Function isDominated(AP = ⟨A, π⟩):
23     if ∃π' ∈ Sols: c(π') ⪯_ε f(AP) then
24         return true
25     if ∃x ∈ G_cl^T(s(AP)) : x ⪯ Tr(g(AP)) then
26         return true
27     return false
28 Function addToOpen(AP = ⟨A, π⟩):
29     for AP' = ⟨A', π'⟩ ∈ Open[s(AP)] do
30         AP_new = ⟨A_new, π_new⟩ ← merge(AP, AP')
31         if AP_new is ε-bounded then
32             remove AP' from Open
33             add AP_new to Open
34             return
35     add AP to Open
36     return
```

Let $Open[s]$ be the set of apex-path pairs in $Open$ that contains state s. Whenever A*pex attempts to add an apex-path pair $\mathcal{AP}$ to $Open$, A*pex first tries to *merge* $\mathcal{AP}$ with another apex-path pair in $Open[s(\mathcal{AP})]$ if the resulting apex-path pair is ε-bounded (Lines 29-34). When merging two apex-path pairs, the new apex resulting from merging two apex-path pairs is the component-wise minimum of the apexes of the two apex-path pairs, and the new representative path is either one of the two representative paths of the two apex-path pairs. Zhang et al. (2022a) proposed several heuristics for choosing the new representative path.

A*pex builds upon PP-A* (Goldin and Salzman 2021) and obtains average speed-ups of more than two times over it. PP-A* is an approximate bi-objective search algorithm that, in turn, has been shown to obtain average speed-ups of up to an order of magnitude over BOA*-ε. Compared to BOA*-ε, A*pex further reduce the search effort by merg-

Algorithm 2: A-A*pex

```
   Input  : P = ⟨G, s_start, s_goal⟩, getNextEps(), h
38 Pruned ← {[s_start]}
39 Sols ← ∅
40 while Search not halted do
41     ε_curr ← getNextEps()
42     Open ← ∅
43     Pruned′ ← Pruned; Pruned ← ∅
44     for each π ∈ Pruned′ do
45         addToOpen(⟨c(π), π⟩)
46     findApproxPF(ε_curr)
47     if Pruned = ∅ then
48         break
49 return Sols
50 Function findApproxPF(ε):
       /* Same as Lines 5 to 21 in
          Algorithm 1                          */
51 Function isDominated(AP = ⟨A, π⟩):
52     if ∃π′ ∈ Sols: c(π′) ⪯_ε_curr f(AP) then
+53         if not c(π′) ⪯ c(π) + h(s(AP)) then
+54             add π to Pruned
55         return true
56     if ∃x ∈ G_cl^T(s(AP)) : x ⪯ Tr(g(AP)) then
+57         π′ ← the representative path of the apex-path pair
               corresponding to x
+58         if not c(π′) ⪯ c(π) then
+59             add π to Pruned
60         return true
61     return false
62 Function addToOpen(AP = ⟨A, π⟩):
63     for AP′ = ⟨A′, π′⟩ ∈ Open[s(AP)] do
64         AP_new = ⟨A_new, π_new⟩ ← merge(AP, AP′)
65         if AP_new is ε_curr-bounded then
66             remove AP′ from Open
67             add AP_new to Open
+68             π_pruned ← π′ if π = π_new or π otherwise
+69             if not c(π_new) ⪯ c(π_pruned) then
+70                 add π_pruned to Pruned
71             return
72     add AP to Open
73     return
```

ing search nodes. This motivates us to investigate anytime multi-objective search algorithms that build upon A*pex.

4 A-A*pex

In this section, we describe A-A*pex. A-A*pex calls A*pex repeatedly with smaller and smaller ε-values to compute better and better approximate frontiers. From one iteration to the next, A-A*pex can either reuse its previous search effort or restart the search from scratch. We first describe the variant of A-A*pex that reuses its previous search effort. Then, we describe the variant that restarts the search from scratch, which only differs in a few lines of pseudo-code.

The representative paths A*pex discards when pruning or merging apex-path pairs might still be extendable to Pareto-optimal solutions. One can store such representative paths and resume searching on them later. For example, consider the case when A*pex prunes an apex-path pair $\mathcal{AP}$ because its $\mathbf{f}$-value is ε-dominated by some solution π' in $Sols$ (Line 24). The representative path π of $\mathcal{AP}$ might still be extendable to a Pareto-optimal solution if $\mathbf{c}(\pi) + \mathbf{h}(s(\mathcal{AP}))$ (which weakly dominates the cost of any solution extending π) is not weakly dominated by $\mathbf{c}(\pi')$. Similar observations hold for the representative paths of the apex-path pairs pruned on Line 26 because of Condition 1 and the representative paths that are not chosen as new representative paths when merging apex-path pairs on Line 30. Our technique for reusing previous search effort is based on these observations.

Algorithm 2 shows the pseudo-code of the variant of A-A*pex that reuses its previous search effort. The input to A-A*pex is a problem instance, an approximation factor update scheme encoded by the getNextEps function, and a heuristic $\mathbf{h}$. A-A*pex maintains a list $Pruned$ of pruned paths, which is initialized with path $[s_{\text{start}}]$ (Line 38), and a set $Sols$ of solutions. In each iteration of its main loop (Lines 40-48), A-A*pex first calls getNextEps to decrease the *current approximation factor* $\varepsilon_{\text{curr}}$ (Line 41). It then initializes $Open$ with the paths in $Pruned$ (Lines 42-45): A-A*pex first moves the paths from $Pruned$ to another set $Pruned'$ (Line 43) and then calls addToOpen with each of these paths. Some of these paths might be put back into $Pruned$ by addToOpen, which we will explain later. A-A*pex then calls findApproxPF to compute an $\varepsilon_{\text{curr}}$-approximate frontier (Line 46). A-A*pex shares the findApproxPF function with A*pex. However, its isDominated and addToOpen functions are modified (Lines 51-73). We use "+" before the line numbers to indicate the changes:

1. **Lines 53-54:** If the $\mathbf{f}$-value of an apex-path pair $\mathcal{AP}$ is $\varepsilon_{\text{curr}}$-dominated by the cost of some solution π' in $Sols$ but the representative path π of $\mathcal{AP}$ satisfies that $\mathbf{c}(\pi) + \mathbf{h}(s(\mathcal{AP}))$ is not weakly dominated by $\mathbf{c}(\pi')$, A-A*pex adds π to $Pruned$.
2. **Lines 57-59:** For each vector $\mathbf{x}$ in $G_{\text{cl}}^T(s)$ for any state s, A-A*pex maintains the representative path of the apex-path pair whose truncated $\mathbf{g}$-value equals $\mathbf{x}$ and resulted in $\mathbf{x}$ being added to $G_{\text{cl}}^T(s)$. If the truncated $\mathbf{g}$-value of an apex-path pair $\mathcal{AP}$ is weakly dominated by some vector $\mathbf{x}$ in $G_{\text{cl}}^T(s(\mathcal{AP}))$ but the representative path π of $\mathcal{AP}$ is not weakly dominated by the representative path of the apex-path pair corresponding to $\mathbf{x}$, A-A*pex adds π to $Pruned$.
3. **Lines 68-70:** When merging two apex-path pairs, one of their representative paths is chosen as the new representative path. Let π_{new} denote the chosen representative path and π_{pruned} denote the other. If π_{pruned} is not weakly dominated by π_{new}, A-A*pex adds π_{pruned} to $Pruned$.

In findApproxPF, A-A*pex does not reuse the truncated $\mathbf{g}$-values in G_{cl}^T from previous iterations because, even if the truncated $\mathbf{g}$-value of an apex-path pair $\mathcal{AP}$ is weakly dominated by a vector $\mathbf{x}$ in G_{cl}^T from previous iterations, $\mathbf{g}(AP)$ is not necessarily weakly dominated by the $\mathbf{g}$-value corresponding to $\mathbf{x}$. The $\varepsilon_{\text{curr}}$-value of the current iteration is also different from those of previous iterations. Thus, A-A*pex can expand apex-path pairs whose $\mathbf{g}$-values are weakly dominated by the $\mathbf{g}$-values of some expanded apex-path pairs containing the same states from previous iterations. How-

ever, this does not affect it computing an $\varepsilon_{\text{curr}}$-approximate frontier in each iteration because it still considers the solutions in $Sols$ from previous iterations in dominance checks. We formally prove its correctness in Theorem 1. When $Pruned$ becomes empty, A-A*pex returns $Sols$ as a Pareto frontier (Line 49).

A-A*pex with restarting. Instead of reusing its previous search effort, A-A*pex can also restart the search from scratch in each iteration. This requires changes only to Lines 42-45, where A-A*pex now initializes $Open$ with path $[s_{\text{start}}]$ instead of the paths in $Pruned$.

Enhanced dominance checks. Although A-A*pex does not reuse the truncated **g**-values from previous iterations for dominance checks, it can still prune an apex-path pair $\mathcal{AP}$ if $\mathbf{g}(\mathcal{AP})$ is weakly dominated by the cost of the representative path π' of an apex-path pair that was expanded in previous iterations and contains the same state as $\mathcal{AP}$. In such a case, the entire set of paths that $\mathcal{AP}$ corresponds to are weakly dominated by π' despite the $\varepsilon_{\text{curr}}$-value. One can thus enhance the dominance checks of A-A*pex by maintaining the set of undominated costs $C(s)$ of the representative paths of all expanded apex-path pairs for each state s and using these for dominance checks. This requires changes only inside the isDominated function to check if the **g**-value of the input apex-path pair $\mathcal{AP}$ is weakly dominated by any vector in $C(s(\mathcal{AP}))$ and adding one line after Line 12 to update $C(s(\mathcal{AP}))$ before expanding $\mathcal{AP}$.

Mix reusing search effort and restarting from scratch. As $\varepsilon_{\text{curr}}$ decreases, findApproxPF often terminates with more expanded nodes and fewer paths in $Pruned$. Hence, restarting from scratch becomes less efficient than using $Pruned$ to initialize $Open$. Let $\#_{\text{exp}}$ and $\#_{\text{pruned}}$ denote the numbers of expanded nodes and pruned paths, respectively. We propose a variant of A-A*pex, called A-A*pex-hybrid, that first restarts the search from scratch in each iteration and starts reusing its search effort when the ratio of $\#_{\text{exp}}$ and $\#_{\text{pruned}}$ of the previous iteration is larger than a threshold. It then keeps reusing its search effort until it terminates. In the experiments, we set the threshold to five empirically based on our preliminary results.

4.1 Theoretical Results

This section provides theoretical results for A-A*pex. We study only the variant of A-A*pex that reuses its previous search effort because all theorems in this section trivially hold for the variant of A-A*pex that restarts the search from scratch. Theorem 1 shows that A-A*pex computes an $\varepsilon_{\text{curr}}$-approximate frontier in each iteration. Theorem 2 shows that A-A*pex eventually computes a Pareto frontier.

Given a solution $\pi_{\text{sol}} = [s_1(=s_{\text{start}}), s_2 \dots s_L(=s_{\text{goal}})]$, we use $\pi_{\text{sol}}^{(l)}, l = 1, 2 \dots L$, to denote its prefix $[s_1, s_2 \dots s_l]$ of π_{sol}. We define a path π to be *l-compatible* with π_{sol} iff (i) the last state of π is s_l and (ii) $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi_{\text{sol}}^{(l)})$. We define a path π to be *compatible* with π_{sol} iff there exists an l for which π is l-compatible with π_{sol}. Thus, path $[s_{\text{start}}]$ is both 1-compatible and compatible with any solution.

Lemma 1. *Consider any solution $\pi_{sol} = [s_1, s_2 \dots s_L]$. If* findApproxPF *expands (that is, reaches Line 13 with) an apex-path pair $\mathcal{AP}$ whose representative path is l-compatible with π_{sol} for some l, $0 \le l < L$, then there exists, when* findApproxPF *terminates, (Case 1) a path that is compatible with π_{sol} in $Pruned$ or (Case 2) a solution in $Sols$ that weakly dominates π_{sol}.*

Proof. We prove this lemma by induction on l, starting from $l = L$ and going backward. Consider the case where findApproxPF expands an apex-path pair $\mathcal{AP}$ whose representative path π is L-compatible with π_{sol}. π is a path to s_L and $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi_{\text{sol}}^{(L)}) = \mathbf{c}(\pi_{\text{sol}})$ because π is L-compatible with π_{sol}. findApproxPF then adds π to $Sols$. There must exist a solution in $Sols$ that weakly dominates π_{sol} when findApproxPF terminates because it only removes a solution from $Sols$ when adding another solution that weakly dominates it (Lines 14-15). Case 2 holds.

Assume the lemma holds for $l+1$ and consider the case where findApproxPF expands an apex-path pair $\mathcal{AP}$ whose representative path π is l-compatible with π_{sol}. Consider the child apex-path pair $\mathcal{AP}' = \langle \mathbf{A}', \pi' \rangle$ of $\mathcal{AP}$ that findApproxPF generates for state s_{l+1} when reaching Line 18. π' weakly dominates $\pi_{\text{sol}}^{(l+1)}$ because $\mathbf{c}(\pi') = \mathbf{c}(\pi) + \mathbf{c}(\langle s_l, s_{l+1} \rangle) \preceq \mathbf{c}(\pi_{\text{sol}}^{(l)}) + \mathbf{c}(\langle s_l, s_{l+1} \rangle) = \mathbf{c}(\pi_{\text{sol}}^{(l+1)})$. We distinguish the following two cases:

1. findApproxPF prunes $\mathcal{AP}'$ on Line 11 or 20 because of the condition on Line 55 or 60. If π' is added to $Pruned$, Case 1 holds. If not and isDominated reaches Line 55 without adding π' to $Pruned$, then there exists a solution in $Sols$ whose cost weakly dominates $\mathbf{c}(\pi') + \mathbf{h}(s_{l+1})$ (which in turn weakly dominates $\mathbf{c}(\pi_{\text{sol}})$ because $\mathbf{h}$ is consistent). Case 2 holds. If isDominated reaches Line 60 without adding π' to $Pruned$, then there exists an expanded apex-path pair (namely, the one mentioned on Line 57) that contains state s_{l+1} and whose representative path weakly dominates π' (and hence is $(l+1)$-compatible with π_{sol}). Because the lemma holds for $l+1$, it thus also holds for l.
2. findApproxPF calls addToOpen with $\mathcal{AP}'$. The algorithm might merge $\mathcal{AP}'$ with other apex-path pairs before extracting the resulting apex-path pair of these merges, denoted as $\mathcal{AP}''$, from $Open$. During these merges, a representative path is completely discarded (i.e., neither chosen as the new representative path nor added to $Pruned$) only if it is weakly dominated by the other representative path. Therefore, if no path that weakly dominates π' is added to $Pruned$ during these merges, the representative path of $\mathcal{AP}''$ must weakly dominate π' (and hence is $(l+1)$-compatible with π_{sol}). If $\mathcal{AP}''$ is pruned on Line 11, the lemma holds as we have already proved. Otherwise, $\mathcal{AP}''$ is expanded, and the lemma holds for l. □

Lemma 2. *Consider any solution π_{sol}. When A-A*pex reaches Line 41, there exists a path compatible with π_{sol} in $Pruned$ or a solution in $Sols$ that weakly dominates π_{sol}.*

Proof. We prove this lemma by induction. When A-A*pex reaches Line 41 for the first time, path $[s_{\text{start}}]$ in $Pruned$ is compatible with π_{sol}, and hence the lemma holds. Assume

that A-A*pex reaches Line 41 and the lemma has held so far. If there exists a solution in $Sols$ that weakly dominates π_{sol}, there must exist a solution in $Sols$ that weakly dominates π_{sol} when A-A*pex reaches Line 41 again because A-A*pex only removes a solution from $Sols$ when adding another solution that weakly dominates it (Lines 14-15). Otherwise, there exists a path π' in $Pruned$ that is compatible with π_{sol}. A-A*pex then calls addToOpen with apex-path pair $\langle \mathbf{c}(\pi'), \pi' \rangle$ on Line 45 and might merge it with other apex-path pairs before findApproxPF extracts the resulting apex-path pair $\mathcal{AP}''$ from $Open$. As we have already proved, if the algorithm does not add a path that weakly dominates π' to $Pruned$ during these merges, the representative path π'' of $\mathcal{AP}''$ must weakly dominate π' (and hence is compatible with π_{sol}). If $\mathcal{AP}''$ is pruned on Line 11, we can distinguish the following cases:

1. The representative path of $\mathcal{AP}''$ is added to $Pruned$. The lemma holds.
2. findApproxPF reaches Line 55 without adding the representative path of $\mathcal{AP}''$ to $Pruned$. There exists a solution in $Sols$ whose cost weakly dominates $\mathbf{c}(\pi'') + \mathbf{h}(s(\mathcal{AP}''))$, which in turn weakly dominates $\mathbf{c}(\pi_{\text{sol}})$ because π'' is compatible with π_{sol}. The lemma holds.
3. findApproxPF reaches Line 60 without adding the representative path of $\mathcal{AP}''$ to $Pruned$. There exists an expanded apex-path pair whose representative path is compatible with π_{sol}. From Lemma 1, the lemma holds.

Otherwise, $\mathcal{AP}''$ is expanded. From Lemma 1, the lemma holds. □

Theorem 1. *Consider any solution π_{sol}. There exists, when A-A*pex reaches the end of Line 46, a solution in $Sols$ that ε_{curr}-dominates solution π_{sol}.*

The proof of Theorem 1 is in the extended version of this paper.[1] It is similar to the proof of Theorem 1 by Zhang et al. (2022a) but uses a different initialization of $Open$.

Theorem 2. *A-A*pex terminates when ε_{curr} becomes sufficiently small, and $Sols$ is then a cost-unique Pareto frontier.*

Proof. Let $Sols_0$ and ε_0 denote $Sols$ after the first run of findApproxPF and its corresponding $\varepsilon_{\text{curr}}$-value, respectively. From Theorem 1, $Sols_0$ is an ε_0-approximate frontier. Consider the subsequent iterations and a solution $\pi_{\text{sol}} \in Sols_0$. findApproxPF does not expand any apex-path pair whose $\mathbf{f}$-value is weakly dominated by the cost of π_{sol} because of the condition on Line 52. Because findApproxPF generates only ε-bounded apex-path pairs for some ε-value smaller than ε_0 and the graph is finite, there are only a finite number of (representative) paths that findApproxPF can generate. Thus, when $\varepsilon_{\text{curr}}$ becomes sufficiently small, findApproxPF merges two apex-path pairs only if one of the representative paths weakly dominates the other and can only choose this representative path as the new representative path. Therefore, the apex of an apex-path pair is always equal to the cost of its representative path and hence findApproxPF cannot reach Line 59 or 70. When $\varepsilon_{\text{curr}}$ becomes sufficiently small, the condition on Line 52 will hold only when $\mathbf{c}(\pi') \preceq \mathbf{f}(\mathcal{AP})$, and hence findApproxPF cannot reach Line 54. $Pruned$ then stays empty, and A-A*pex terminates. From Lemma 2, for any solution π_{sol}, there exists a solution in $Sols$ that weakly dominates π_{sol}. Because findApproxPF adds a solution to $Sols$ only if it is not ε-dominated (and hence not weakly dominated) by any solution in $Sols$ and removes all solutions that are weakly dominated by it from $Sols$, no solution in $Sols$ weakly dominates each other. Thus, $Sols$ is a cost-unique Pareto frontier. □

5 Experimental Results

In our experimental study, we first compare different variants of A-A*pex. We then compare A-A*pex with state-of-the-art multi-objective search algorithms.

We use two graphs: (1) an empty 48×48 four-neighbor grid and (2) the NY road network from the 9th DIMACS Implementation Challenge,[2] which has 264K states and 730K edges. For the empty grid, we generate each cost component as a random integer from 1 to 10 for up to six objectives. We then randomly generate 100 problem instances. The NY road network has two objectives available in the benchmark, namely travel distance (d) and travel time (t). We use the economic cost (m) (Pulido, Mandow, and Pérez-de-la Cruz 2015), the number of edges (l) (Maristany de las Casas et al. 2023), and a random integer from 1 to 100 (r) (Hernández et al. 2023) as the third, fourth, and fifth objectives, respectively. We use the same 100 problem instances used by Sedeño-Noda and Colebrook (2019) and Ahmadi et al. (2021). Following Hernández et al. (2023), each component $h_i(s)$ of the heuristic $\mathbf{h}(s)$ for state s is the minimum cost needed to reach s_{goal} from s for the ith objective.

We implemented all algorithms in C++[3] and ran all experiments on a MacBook with an M1 Pro chip and 32GB of memory. The runtime limit for solving each instance was five minutes. In A-A*pex, the sequence of ε-values output by getNextEps began with 0.1 and was divided by η after every iteration, where η was a predetermined parameter.

There are problem instances where no algorithm finds the entire Pareto frontier within the runtime limit. When computing the approximation error using Equation 1 in these cases, we use the set of undominated solutions that all algorithms computed as a substitution for Π^*.

5.1 Comparing Different Variants of A-A*pex

We compare different variants of A-A*pex on the first 50 problem instances on the NY road network with three objectives (m-t-d). These variants of A-A*pex are:

1. A-A*pex-reuse always reuses its search effort and is our baseline variant of A-A*pex.
2. A-A*pex-reuse-enh always reuses its search effort and also uses the enhanced dominance checks.
3. A-A*pex-restart always restarts the search from scratch.

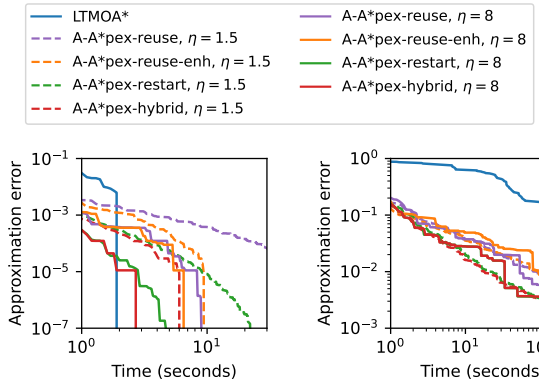
[1] http://idm-lab.org/bib/abstracts/Koen24f.html

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

[3] https://github.com/HanZhang39/MultiObjectiveSearch

	#solved	time (s)	#exp
LTMOA*	47	0.31	331K
A-A*pex-reuse $\eta = 1.5$	35	6.31	2636K
A-A*pex-reuse-enh, $\eta = 1.5$	**40**	1.18	**423K**
A-A*pex-restart, $\eta = 1.5$	38	3.61	5539K
A-A*pex-hybrid, $\eta = 1.5$	**40**	**0.84**	1208K
A-A*pex-reuse $\eta = 2$	36	3.51	1779K
A-A*pex-reuse-enh, $\eta = 2$	40	1.04	**396K**
A-A*pex-restart, $\eta = 2$	39	2.00	3417K
A-A*pex-hybrid, $\eta = 2$	**41**	**0.65**	878K
A-A*pex-reuse $\eta = 4$	38	1.75	1121K
A-A*pex-reuse-enh, $\eta = 4$	40	0.78	**378K**
A-A*pex-restart, $\eta = 4$	40	1.11	1866K
A-A*pex-hybrid, $\eta = 4$	**42**	**0.59**	753K
A-A*pex-reuse $\eta = 8$	39	1.19	908K
A-A*pex-reuse-enh, $\eta = 8$	40	0.81	**367K**
A-A*pex-restart, $\eta = 8$	**42**	0.78	1307K
A-A*pex-hybrid, $\eta = 8$	**42**	**0.41**	515K

Table 1: Results for different algorithms on 50 problem instances on NY with three objectives.



(a) The 35 instances solved by all algorithms.

(b) The other 15 instances.

Figure 1: Approximation error as a function of the runtime for different algorithms on NY with three objectives.

4. A-A*pex-hybrid initially restarts the search from scratch and reuses its search effort in later iterations. It also uses the enhanced dominance checks.

We evaluate each variant of A-A*pex with $\eta \in \{1.5, 2, 4, 8\}$. We also evaluate LTMOA* (Hernández et al. 2023), a state-of-the-art multi-objective search algorithm that generalizes BOA* to more than two objectives.

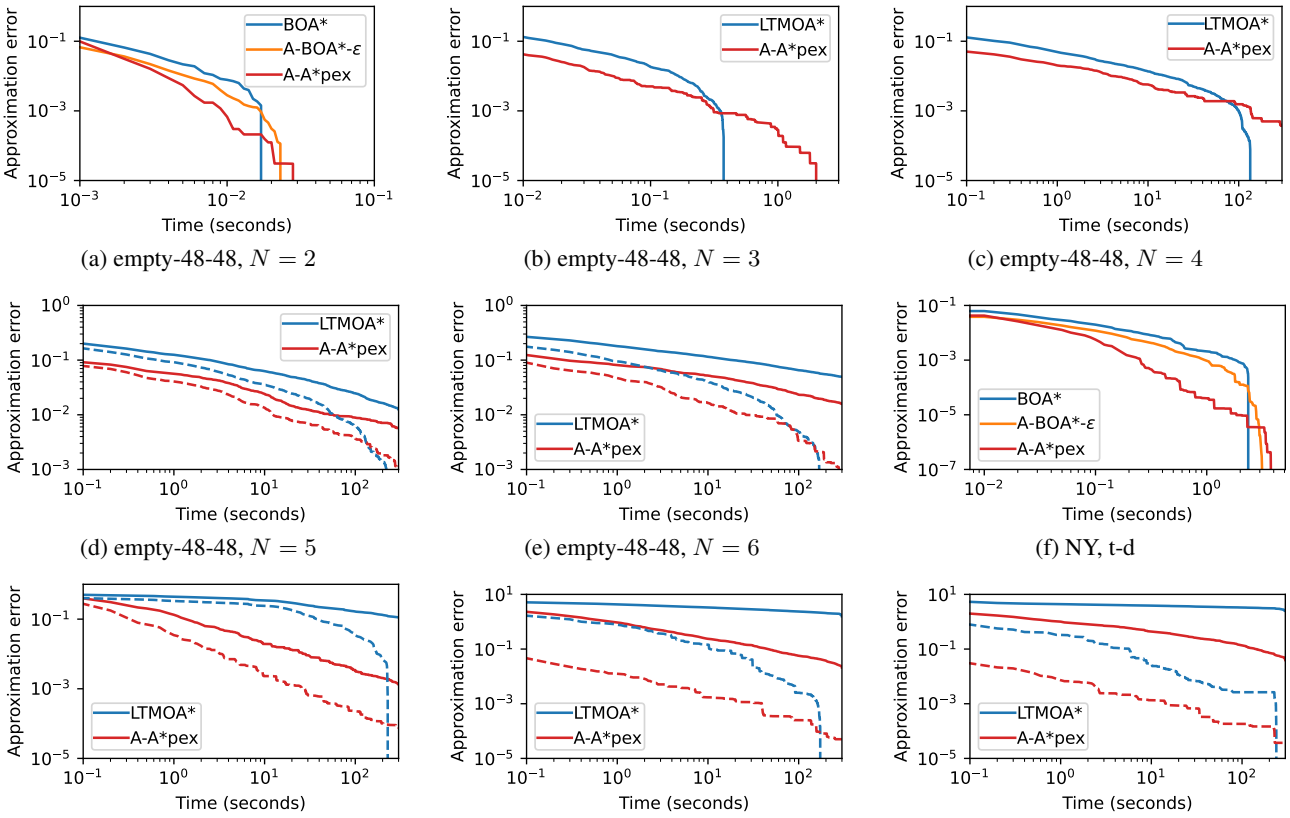
Table 1 shows the numbers of solved problem instances (i.e., the number of instances for which an algorithm finds the entire Pareto frontier within the runtime limit), average runtimes (in seconds), and average numbers of expanded nodes for all algorithms. All averages are over those instances that all algorithms solve. LTMOA* has the largest number of solved instances and the smallest average runtime and number of expanded nodes. The average runtime of each variant of A-A*pex decreases as η increases because larger η result in fewer iterations of A-A*pex. Adding enhanced dominance checks decreases the average runtime of A-A*pex-reuse and results in the smallest node expansions of all A-A*pex variants. In general, A-A*pex-hybrid outperforms the other three A-A*pex variants in terms of the number of solved instances and average runtime.

Figure 1 shows the approximation error as a function of the runtime for LTMOA* and all A-A*pex variants with $\eta = 1.5$ and $\eta = 8$. We use only two η-values to keep the figure clean. We divide the instances into two groups, namely the instances solved by all algorithms (Figure 1(a)) and the other instances (Figure 1(b)). The approximation error of each algorithm is averaged over all instances in a group. A-A*pex-reuse and A-A*pex-reuse-enh have larger approximation errors than A-A*pex-restart and A-A*pex-hybrid in the beginning of the search for both η-values, which shows that restarting the search from scratch is more efficient in the earlier iterations. In Figure 1(a), the approximation error of A-A*pex-reuse, A-A*pex-reuse-enh, and A-A*pex-hybrid quickly drops in the later iterations, which shows that reusing search effort is more efficient in the later iterations. Therefore, by mixing these two techniques, A-A*pex-hybrid often finds the Pareto frontier faster than the other variants of A-A*pex. In Figure 1(b), all variants of A-A*pex have a smaller approximation error than LTMOA* for the entire five minutes of runtime for both η-values. Although we expect LTMOA* to find Pareto frontiers faster than A-A*pex if sufficient runtime is provided, all variants of A-A*pex often compute solution sets with approximation errors smaller than 0.01 faster than LTMOA*.

5.2 Comparing with the State-of-the-Art

We compare the hybrid variant of A-A*pex with $\eta = 4$ with the state-of-the-art multi-objective search algorithms BOA* (Hernández et al. 2023) and A-BOA*-ε (Zhang et al. 2022b) on problem instances with two objectives and LTMOA* on problem instances with more than two objectives.

Figure 2 shows the results for different graphs and combinations of objectives. We use solid lines for all problem instances and dashed lines for those problem instances whose entire Pareto frontiers are computed within the runtime limit. In all cases, A-A*pex reduces the approximation error faster than the other algorithms in the beginning of the search. Because LTMOA* and BOA* compute solutions in lexicographically increasing order of their costs, they can exactly "cover" part of the Pareto frontier while completely missing the rest during most part of the search, which explains their high approximation errors at the beginning. This behavior is undesirable from the perspective of approximating the entire Pareto frontier when a limited deliberation time is given. When a sufficient runtime is provided, LTMOA* and BOA* find the Pareto frontier faster than A-A*pex and hence have smaller approximation errors than A-A*pex. However, this happens only after the approximation error becomes smaller than 0.01, even smaller than 0.001 in many cases, which means that A-A*pex computes 0.01-approximate frontiers faster than BOA* or LTMOA*. For every solution π, there exists a solution π' in a 0.01-approximate frontier such that π is at most 1% worse than π' for any objective, which is

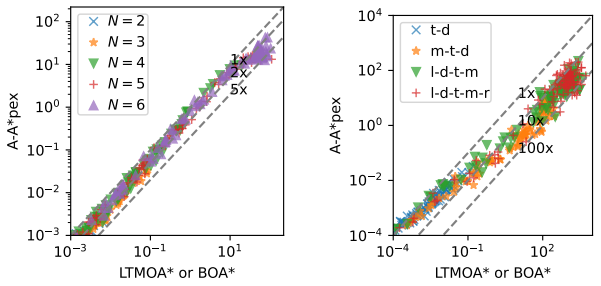


(a) empty-48-48, $N = 2$ (b) empty-48-48, $N = 3$ (c) empty-48-48, $N = 4$

(d) empty-48-48, $N = 5$ (e) empty-48-48, $N = 6$ (f) NY, t-d

(g) NY, m-t-d (h) NY, l-d-t-m (i) NY, l-d-t-m-r

Figure 2: Anytime behaviors of different algorithms on different graphs with different combinations of objectives. Each plot shows the approximation error as a function of the runime for each algorithm over all 100 problem instances (solid line) and over only those problem instances solved by at least one algorithm (dashed line).



(a) empty-48-48 (b) NY

Figure 3: AUCs for LTMOA* (or BOA* for two objectives) and A-A*pex on all problem instances.

sufficient for many real-world problems.

We also compute the *Area Under the Curve* (AUC) of the approximation error for each problem instance P and algorithm A, formally defined as $\text{AUC}_A(P) = \int_0^{t^{\text{limit}}} e(t)$, where t^{limit} is the runtime limit of five minutes and e is the approximation error as a function of the runtime. We compare A-A*pex with LTMOA* (or BOA* for two objectives) as the baseline. Figure 3 shows the results. The numbers along the dashed lines denote how many times the AUC of A-A*pex is smaller than that of the baseline. For instances that are more difficult to solve (the points in the top-right corners), A-A*pex always has a smaller AUC than the baseline. The reduction of the AUCs is larger on the NY road network, where A-A*pex has up to $100\times$ smaller AUCs, than the empty grid. However, A-A*pex still has up to $5\times$ smaller AUCs on the empty grid for some problem instances.

In general, the improvement of the approximation error and the AUC of A-A*pex over other multi-objective search algorithms is more substantial on the NY road network than the empty grid. This is probably because Pareto-optimal solutions likely share more states on the NY road network, which allows A-A*pex to merge more nodes.

6 Conclusions

In this paper, we proposed A-A*pex, an anytime approximate multi-objective search algorithm that builds upon A*pex. Our experimental results showed that A-A*pex often computes approximate frontiers with smaller approximation errors than state-of-the-art multi-objective search algorithms for short deliberation times.

Acknowledgements

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1409987, 1724392, 1817189, 1837779, 1935712, and 2112533. It was also supported by the United States-Israel Binational Science Foundation (BSF) under grant number 2021643, the Ministry of Science & Technology, Israel, under grant number 3-17385, the National Center for Artificial Intelligence CENIA FB210017, Basal ANID, and the Centro Ciencia & Vida FB210008, Financiamiento Basal ANID. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, or governments.

References

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

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

Breugem, T.; Dollevoet, T.; and van den 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 (2nd ed.)*. Springer.

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

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

Goldin, B.; and Salzman, O. 2021. Approximate Bi-Criteria Search by Efficient Representation of Subsets of the Pareto-Optimal Frontier. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 149–158.

Hernández, C.; Yeoh, W.; Baier, J. A.; Felner, A.; Salzman, O.; Zhang, H.; Chan, S.-H.; and Koenig, S. 2023. Multi-objective Search via Lazy and Efficient Dominance Checks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 7223–7230.

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

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. *Advances in Neural Information Processing Systems*.

Maristany de las Casas, P.; Kraus, L.; Sedeño-Noda, A.; and Borndörfer, R. 2023. Targeted Multiobjective Dijkstra Algorithm. *Networks*, 82(3): 277–298.

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

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.

Richter, S.; Thayer, J.; and Ruml, W. 2010. The Joy of Forgetting: Faster Anytime Search via Restarting. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 20, 137–144.

Salzman, O.; Felner, A.; Hernández, C.; Zhang, H.; Chan, S.; and Koenig, S. 2023. Heuristic-Search Approaches for the Multi-Objective Shortest-Path Problem: Progress and Research Opportunities. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 6759–6768.

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

Tsaggouris, G.; and Zaroliagis, C. D. 2009. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-Linear Objectives with Applications. *Theory of Computing Systems*, 45(1): 162–186.

Warburton, A. 1987. Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems. *Operations Research*, 35(1): 70–79.

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

Zhang, H.; Salzman, O.; Kumar, T. K. S.; Felner, A.; Hernández, C.; and Koenig, S. 2022b. Anytime Approximate Bi-Objective Search. In *Symposium on Combinatorial Search (SOCS)*, 199–207.