A*pex: Efficient Approximate Multi-Objective Search on Graphs

Han Zhang[1], **Oren Salzman**[2], **T. K. Satish Kumar**[1], **Ariel Felner**[3],
Carlos Hernández Ulloa[4], **Sven Koenig**[1]

[1] University of Southern California
[2] Technion - Israel Institute of Technology
[3] Ben-Gurion University
[4] Universidad San Sebastian

zhan645@usc.edu, osalzman@cs.technion.ac.il, tkskwork@gmail.com, felner@bgu.ac.il,
carlos.hernandez@uss.cl, skoenig@usc.edu

Abstract

In multi-objective search, edges are annotated with cost vectors consisting of multiple cost components. A path dominates another path with the same start and goal vertices iff the component-wise sum of the cost vectors of the edges of the former path is "less than" the component-wise sum of the cost vectors of the edges of the latter path. The Pareto-optimal solution set is the set of all undominated paths from a given start vertex to a given goal vertex. Its size can be exponential in the size of the graph being searched, which makes multi-objective search time-consuming. In this paper, we therefore study how to find an approximate Pareto-optimal solution set for a user-provided vector of approximation factors. The size of such a solution set can be significantly smaller than the size of the Pareto-optimal solution set, which enables the design of approximate multi-objective search algorithms that are efficient and produce small solution sets. We present such an algorithm in this paper, called A*pex. A*pex builds on PP-A*, a state-of-the-art approximate bi-objective search algorithm (where there are only two cost components) but (1) makes PP-A* more efficient for bi-objective search and (2) generalizes it to multi-objective search for any number of cost components. We first analyze the correctness of A*pex and then experimentally demonstrate its efficiency advantage over existing approximate algorithms for bi- and tri-objective search.

1 Introduction

In multi-objective search, we are given a directed graph whose edges are annotated with cost vectors consisting of multiple cost components. A path π dominates a path π' iff the component-wise sum $c(\pi)$ of the costs of the edges of path π is less than the component-wise sum $c(\pi')$ of the costs of the edges of path π', that is, each cost component of $c(\pi)$ is no larger than the corresponding cost component of $c(\pi')$ and one of them is smaller. The objective of multi-objective search is to find the Pareto-optimal solution set of paths from a given start vertex to a given goal vertex, that is, all undominated paths from the start vertex to the goal vertex. This objective generalizes the one of single-objective search to find all cost-minimal paths from the start vertex to the goal vertex. It 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).

Unfortunately, the size of the Pareto-optimal solution set can be exponential in the size of the graph being searched (Ehrgott 2005; Breugem, Dollevoet, and van den Heuvel 2017), which makes multi-objective search time-consuming. Researchers have therefore proposed to find an approximate Pareto-optimal solution set instead (Breugem, Dollevoet, and van den Heuvel 2017; Tsaggouris and Zaroliagis 2009; Warburton 1987; Goldin and Salzman 2021; Perny and Spanjaard 2008), that is, a set of paths such that any path in the Pareto-optimal solution set is ε-approximately dominated by some path in the approximate Pareto-optimal solution set. A path π ε-approximately dominates a path π' for some $\varepsilon \geq 0$ iff each cost component of $c(\pi)$ is no larger than $(1 + \varepsilon)$ times the corresponding cost component of $c(\pi')$, where ε is a user-provided approximation factor that can be different for different cost components. Different approximate Pareto-optimal solution sets (with different cardinalities) can exist for a given multi-objective search instance and approximation factor, but their cardinalities are typically much smaller than those of the Pareto-optimal solution set even for small approximation factors. This property can be exploited to create approximate multi-objective search algorithms that are efficient and produce small solution sets.

We present such an algorithm in this paper, called A*pex. A*pex is a multi-objective search algorithm that finds an ε-approximate Pareto-optimal solution set for a user-provided approximation factor. It builds on PP-A* (Goldin and Salzman 2021), a state-of-the-art approximate bi-objective search algorithm (where there are only two cost components), but generalizes the representation of sets of paths used by PP-A* to (1) make PP-A* more efficient for bi-objective search and (2) generalize it to multi-objective search for any number of cost components. We first analyze the correctness of A*pex using proof techniques in the proof of Lemma 4 that also apply to PP-A* and then experimentally demonstrate its efficiency advantage over PP-A* for bi-objective search and an approximate baseline algorithm

derived from NAMOA*dr, a state-of-the-art multi-objective search algorithm, for tri-objective search. We obtain average speed-ups of more than two times over PP-A* and more than seven times over the baseline algorithm for a road map with more than 1.5 million vertices and an approximation factor of 0.01.

2 Terminology and Problem Definition

Boldface font denotes vectors or vector functions. v_i denotes the i-th component of vector or vector function $\mathbf{v}$. The addition of two vectors $\mathbf{v}$ and $\mathbf{v}'$ of the same length N is defined as $\mathbf{v} + \mathbf{v}' = [v_1 + v'_1, v_2 + v'_2 \dots v_N + v'_N]$. $\mathbf{v} \preceq \mathbf{v}'$ denotes that $v_i \leq v'_i$ for all $i = 1, 2 \dots N$. In this case, we say that $\mathbf{v}$ *weakly dominates* $\mathbf{v}'$. $\mathbf{v} \prec \mathbf{v}'$ denotes that $\mathbf{v} \preceq \mathbf{v}'$ and there exists an $i \in \{1, 2 \dots N\}$ with $v_i < v'_i$. In this case, we say that $\mathbf{v}$ *dominates* $\mathbf{v}'$. $\mathbf{v} \preceq_{\boldsymbol{\varepsilon}} \mathbf{v}'$ for an *approximation factor* (or, more precisely, vector of approximation factors) $\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2 \dots \varepsilon_N] \in \mathbb{R}^N_{\geq 0}$ denotes that $v_i \leq (1 + \varepsilon_i)v'_i$ for all $i = 1, 2 \dots N$. In this case, we say that $\mathbf{v}$ *$\boldsymbol{\varepsilon}$-dominates* $\mathbf{v}'$. The *truncate function* Tr takes a vector as input and outputs the vector with its first component deleted.

A *(multi-objective search) graph* is a tuple $\langle S, E, \mathbf{c} \rangle$, where S is a finite set of *states* and $E \subseteq S \times S$ is a finite set of *edges*. $succ(s) = \{s' \in S : \langle s, s' \rangle \in E\}$ denotes the successors of state s. Cost function $\mathbf{c} : E \to \mathbb{R}^N_{\geq 0}$ maps an edge to its cost, which is a vector of its N cost components. The graph is called bi-objective in case $N = 2$ and tri-objective in case $N = 3$.

A *(multi-objective search) instance* is a tuple $P = \langle S, E, \mathbf{c}, s_{\text{start}}, s_{\text{goal}} \rangle$, where $\langle S, E, \mathbf{c} \rangle$ is a graph whose edges have N cost components, $s_{\text{start}} \in S$ is the start state, and $s_{\text{goal}} \in S$ is the goal state. The instance is called bi-objective in case $N = 2$ and tri-objective in case $N = 3$.

A *path* from state s_1 to state s_ℓ is a sequence of states $\pi = [s_1, s_2 \dots s_\ell]$ with $\langle s_j, s_{j+1} \rangle \in E$ for all $j = 1, 2 \dots \ell - 1$. $s_1 = s_{\text{start}}$ unless mentioned otherwise. $\mathbf{c}(\pi) = \sum_{j=1}^{\ell-1} \mathbf{c}(\langle s_j, s_{j+1} \rangle)$ denotes the cost of path π. The path is a *solution* iff it is from s_{start} to s_{goal}. It can be *extended* with an edge $\langle s_\ell, s_{\ell+1} \rangle$ to obtain path $[s_1, s_2 \dots s_\ell, s_{\ell+1}]$. It *dominates (resp. weakly dominates)* a path π' iff $\mathbf{c}(\pi) \prec \mathbf{c}(\pi')$ (resp. $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi')$). A *Pareto-optimal* solution for an instance P is a solution that is not dominated by any solution of P. The *Pareto-optimal solution set* is the set of all Pareto-optimal solutions. $\pi \preceq_{\boldsymbol{\varepsilon}} \pi'$ denotes that $c(\pi) \preceq_{\boldsymbol{\varepsilon}} c(\pi')$. In this case, we say that path π *$\boldsymbol{\varepsilon}$-dominates* path π'. An *$\boldsymbol{\varepsilon}$-approximate Pareto-optimal solution set* is a set of solutions such that, for any Pareto-optimal solution π', there exists a solution π in the $\boldsymbol{\varepsilon}$-approximate Pareto-optimal solution set with $\pi \preceq_{\boldsymbol{\varepsilon}} \pi'$. The Pareto-optimal solution set is a $\mathbf{0}$-approximate Pareto-optimal solution set but not necessarily vice versa. For example, the set of all solutions is also a $\mathbf{0}$-approximate Pareto-optimal solution set. Our objective is to find an explicit representation of an $\boldsymbol{\varepsilon}$-approximate Pareto-optimal solution set for a given instance quickly, ideally one of small size. A multi-objective search algorithm that finds an $\boldsymbol{\varepsilon}$-approximate Pareto-optimal solution set is also called an ($\boldsymbol{\varepsilon}$-)approximate multi-objective search algorithm.

A heuristic function $\mathbf{h} : S \to \mathbb{R}^N_{\geq 0}$ provides a lower bound on the cost of any path from any given state s to the goal state. We assume that the provided heuristic function $\mathbf{h}$ 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 best-first multi-objective search framework since all state-of-the-art multi-objective search algorithms, including our new one, are based on it.

A best-first multi-objective search algorithm computes a (usually Pareto-optimal or $\boldsymbol{\varepsilon}$-approximate Pareto-optimal) solution set by maintaining a priority queue $Open$, which contains the generated but not yet expanded nodes. Each node n contains a state $s(n)$ and a $\mathbf{g}$-value $\mathbf{g}(n)$. We define an $\mathbf{f}$-value for the node as $\mathbf{f}(n) = \mathbf{g}(n) + \mathbf{h}(s(n))$. $Open$ is initialized with a node that contains the start state s_{start} and the $\mathbf{g}$-value $\mathbf{0}$. At each iteration, the algorithm extracts a node from $Open$ with the smallest $\mathbf{f}$-value of all nodes in $Open$ and performs a *dominance check* to determine whether the node or any of its descendants have the potential to be in the solution set. If not, it discards the extracted node. If so and the node contains the goal state, it adds the node to the solution set. Otherwise, it expands the node by generating a new node for each of the successors of the state contained in the node. The algorithm performs dominance checks for each generated node to determine whether the generated node or any of its descendants have the potential to be in the solution set. If not, it discards the generated node. Otherwise, it adds the generated node to $Open$. When $Open$ becomes empty, the algorithm terminates and returns the solution set.

Different multi-objective search algorithms, such as NAMOA* (Mandow and De La Cruz 2010), NAMOA*dr (Pulido, Mandow, and Pérez-de-la Cruz 2015), BOA* (Hernandez et al. 2020), and PP-A* (Goldin and Salzman 2021), conform to this framework but differ in which information is contained in the nodes, which node is extracted from $Open$, and how the dominance checks work.

3.1 NAMOA* and NAMOA*dr

NAMOA* is a best-first multi-objective search algorithm that finds the Pareto-optimal solution set. Each node in $Open$ corresponds to a path π. The path contains the state $s(\pi)$ that is the last state in path π and the $\mathbf{g}$-value $\mathbf{g}(\pi) = \mathbf{c}(\pi)$ that is the cost of path π.

NAMOA* maintains two sets of $\mathbf{f}$-values for each state s, namely, the closed set $G_{\text{cl}}(s)$, that contains the $\mathbf{f}$-values of all expanded paths that contain state s, and the open set $G_{\text{op}}(s)$, that contains the $\mathbf{f}$-values of all generated but not yet expanded paths that contain state s.[1] At each iteration, NAMOA* extracts a path π from $Open$ whose $\mathbf{f}$-value is undominated by the $\mathbf{f}$-value of any other path in $Open$ and adds $\mathbf{f}(\pi)$ to $G_{\text{cl}}(s(\pi))$. If state $s(\pi)$ is the goal state, then NAMOA* adds the path to the solution set and removes all paths from $Open$ whose $\mathbf{f}$-values are dominated by $\mathbf{f}(\pi)$.

[1] The notation $G_{\text{cl}}(s)$ and $G_{\text{op}}(s)$ is used here for historical reasons since these sets originally contained $\mathbf{g}$-values. We use $\mathbf{f}$-values instead since it simplifies the descriptions of the search algorithms.

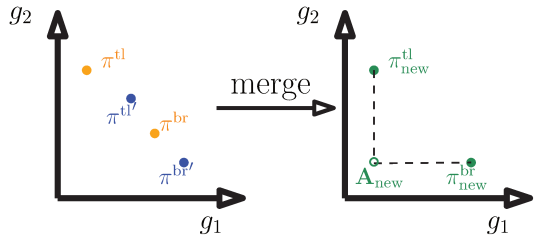

Figure 1: An example, adapted from (Goldin and Salzman 2021), of merging path pairs $\langle \pi^{\mathrm{tl}}, \pi^{\mathrm{br}} \rangle$ (orange) and $\langle \pi^{\mathrm{tl}\prime}, \pi^{\mathrm{br}\prime} \rangle$ (blue) into path pair $\langle \pi^{\mathrm{tl}}_{\mathrm{new}}, \pi^{\mathrm{br}}_{\mathrm{new}} \rangle$ (green).

Otherwise, for each edge $\langle s(\pi), s' \rangle \in E$, NAMOA* extends path π with the edge to obtain an extended path π'. It discards the extended path if its $\mathbf{f}$-value is dominated by an $\mathbf{f}$-value in $G_{\mathrm{cl}}(s') \cup G_{\mathrm{op}}(s') \cup G_{\mathrm{cl}}(s_{\mathrm{goal}})$. Otherwise, it removes all paths that contain state s' and whose $\mathbf{f}$-values are dominated by $\mathbf{f}(\pi')$ from $Open$ (and removes their $\mathbf{f}$-values from $G_{\mathrm{op}}(s')$) and then adds the extended path to $Open$ (and adds its $\mathbf{f}$-value to $G_{\mathrm{op}}(s')$).

NAMOA*dr improves on NAMOA* with respect to the efficiency of dominance checking, which can be the computational bottleneck of multi-objective search algorithms (Pulido, Mandow, and Pérez-de-la Cruz 2015). NAMOA*dr always extracts a path from $Open$ with the lexicographically smallest $\mathbf{f}$-value of all paths in $Open$. When NAMOA*dr generates a path π, it therefore holds that $f_1(\pi) \geq \max\{f_1(\pi') : f_1(\pi') \in G_{\mathrm{cl}}(s(\pi))\}$ and $f_1(\pi) \geq \max\{f_1(\pi') : f_1(\pi') \in G_{\mathrm{cl}}(s_{\mathrm{goal}})\}$ since the heuristic function is consistent. Thus, NAMOA*dr does not need to check the first component when checking whether the $\mathbf{f}$-value of path π is dominated by an $\mathbf{f}$-value in $G_{\mathrm{cl}}(s(\pi)) \cup G_{\mathrm{cl}}(s_{\mathrm{goal}})$. Instead of maintaining the set $G_{\mathrm{cl}}(s)$ of $\mathbf{f}$-values for each state s, NAMOA*dr therefore maintains only the often significantly smaller set $G^T_{\mathrm{cl}}(s)$ of undominated truncated $\mathbf{f}$-values.

When $Open$ becomes empty, NAMOA* and NAMOA*dr terminate and return the solution set as the Pareto-optimal solution set.

Perny and Spanjaard (2008) suggest to compute an ε-approximate Pareto-optimal solution set by discarding an extracted or extended path if its $\mathbf{f}$-value is ε-dominated by the $\mathbf{f}$-value of some path in the solution set. NAMOA*dr-ε is our best-first multi-objective search algorithm that makes this change to NAMOA*dr to find an ε-approximate Pareto-optimal solution set.

3.2 PP-A*

PP-A* (Goldin and Salzman 2021) is a best-first bi-objective search algorithm that finds an ε-approximate Pareto-optimal solution set for a user-provided approximation factor ε. Each node in $Open$ corresponds to an ε-bounded *path pair* $\mathcal{PP} = \langle \pi^{\mathrm{tl}}, \pi^{\mathrm{br}} \rangle$ (where "tl" and "br" stand for "top-left" and "bottom-right," respectively) with $s(\pi^{\mathrm{tl}}) = s(\pi^{\mathrm{br}})$, $g_1(\pi^{\mathrm{tl}}) \leq g_1(\pi^{\mathrm{br}})$, and $g_2(\pi^{\mathrm{tl}}) \geq g_2(\pi^{\mathrm{br}})$. The path pair contains state $s(\mathcal{PP}) = s(\pi^{\mathrm{br}})$ and g-value (also called apex) $\mathbf{g}(\mathcal{PP}) = [g_1(\pi^{\mathrm{tl}}), g_2(\pi^{\mathrm{br}})]$. It is *$\varepsilon$-bounded* iff $g_1(\pi^{\mathrm{br}}) \leq (1+\varepsilon_1) g_1(\pi^{\mathrm{tl}})$ and $g_2(\pi^{\mathrm{tl}}) \leq (1+\varepsilon_2) g_2(\pi^{\mathrm{br}})$ or, equivalently, iff the g-values of both paths π^{tl} and π^{br} ε-dominate the g-value of path pair $\mathcal{PP}$.

Algorithm 1: PP-A*

Input : $P = \langle S, E, \boldsymbol{c}, s_{\mathrm{start}}, s_{\mathrm{goal}} \rangle$
$\boldsymbol{h}$
$\boldsymbol{\varepsilon} = [\varepsilon_1, \varepsilon_2]$

$Open \leftarrow \{\langle [s_{\mathrm{start}}], [s_{\mathrm{start}}] \rangle\}$
solutions $\leftarrow \{\}$
foreach $s \in S$ **do**
  $f_2^{\min}(s) \leftarrow \infty$
while $Open \neq \emptyset$ **do**
  $\mathcal{PP} = \langle \pi^{\mathrm{tl}}, \pi^{\mathrm{br}} \rangle \leftarrow Open$.extract_min()
  if *is_dominated($\mathcal{PP}$)* **then**
    continue
  $f_2^{\min}(s(\mathcal{PP})) \leftarrow f_2(\mathcal{PP})$
  if $s(\mathcal{PP}) = s_{goal}$ **then**
    insert($\mathcal{PP}$, solutions)
    continue
  for $s' \in succ(s(\mathcal{PP}))$ **do**
    $\mathcal{PP}' \leftarrow \langle$extend($\pi_{\mathrm{tl}}, \langle s(\mathcal{PP}), s' \rangle$), extend($\pi_{\mathrm{br}}, \langle s(\mathcal{PP}), s' \rangle$)$\rangle$
    if *is_dominated($\mathcal{PP}'$)* **then**
      continue
    insert($\mathcal{PP}'$, $Open$)
return $\{\pi^{br} : \langle \pi^{tl}, \pi^{br} \rangle \in solutions\}$

Function *is_dominated($\mathcal{PP}$)*:
  if $f_2^{min}(s_{goal}) \leq (1+\varepsilon_2) f_2(\mathcal{PP})$ **then**
    return *true*
  if $f_2^{min}(s(\mathcal{PP})) \leq f_2(\mathcal{PP})$ **then**
    return *true*
  return *false*

Function *insert($\mathcal{PP}$, list)*:
  for $\mathcal{PP}' \in list$ **do**
    $\mathcal{PP}_{\mathrm{new}} \leftarrow merge(\mathcal{PP}, \mathcal{PP}')$
    if $\mathcal{PP}_{new}$ *is ε-bounded* **then**
      remove $\mathcal{PP}'$ from *list*
      add $\mathcal{PP}_{\mathrm{new}}$ to *list*
      return
  add $\mathcal{PP}$ to *list*
  return

While NAMOA* reasons about single paths, PP-A* represents sets of paths with the same last state and similar g-values as ε-bounded path pairs, which results in small numbers of path pair expansions and thus small runtimes. For ε-bounded path pair $\langle \pi^{\mathrm{tl}}, \pi^{\mathrm{br}} \rangle$, the g-value $\mathbf{g}(\pi^{\mathrm{tl}}) = [g_1(\pi^{\mathrm{tl}}), g_2(\pi^{\mathrm{tl}})]$ of top-left path π^{tl} is the lexicographically smallest g-value of all paths in this set, and the vector $[g_2(\pi^{\mathrm{br}}), g_1(\pi^{\mathrm{br}})]$ of bottom-right path π^{br} (called its *reverse* g-*value*) is the lexicographically smallest such vector of all paths in this set. Also, the g-value of the path pair weakly dominates the g-values of all paths in this set (because $g_1(\pi^{\mathrm{tl}})$ is the smallest g_1-value of all paths in this set and $g_2(\pi^{\mathrm{br}})$ is the smallest g_2-value of them), and the g-values of both the top-left and bottom-right paths ε-dominate the g-values of all paths in this set (Goldin and Salzman 2021).

Any two path pairs that contain the same state can be

merged into a single path pair, where the top-left path of the merged path pair is the one of the top-left paths of the two path pairs with the lexicographically smaller $\mathbf{g}$-value and the bottom-right path of the merged path pair is the one of the bottom-right paths of the two path pairs with the lexicographically smaller reverse $\mathbf{g}$-value. See Figure 1 for a visualization of the outcome.

Algorithm 1 shows the pseudocode of PP-A*. It starts with a single path pair $\langle [s_{\text{start}}], [s_{\text{start}}] \rangle$ in $Open$ (Line 1). At each iteration, PP-A* extracts a path pair from $Open$ with the lexicographically smallest $\mathbf{f}$-value (Line 6). Since PP-A* is a bi-objective search algorithm, the truncated $\mathbf{f}$-values correspond to and thus are represented by single numbers. Furthermore, PP-A* maintains a set of undominated truncated $\mathbf{f}$-values of all expanded path pairs that contain state s for each state s. This set can be represented by the smallest f_2-value $f_2^{\min}(s)$ of the already expanded path pairs that contain state s, which is used for dominance checking as follows. Both after extracting (that is, after Line 6) and before generating (that is, before Line 17) a path pair $\mathcal{PP}$ that contains state s, PP-A* discards the path pair

1. if there exists an expanded path pair (that is, one that reaches Line 9) that contains the goal state and whose $\mathbf{f}$-value ε-dominates the $\mathbf{f}$-value of path pair $\mathcal{PP}$ (Line 20).
2. if there exists an expanded path pair that contains state s and whose $\mathbf{f}$-value weakly dominates the $\mathbf{f}$-value of path pair $\mathcal{PP}$ (Line 22).

When PP-A* expands a path pair $\mathcal{PP}$ with state s, it generates a child path pair for each successor state s' of state s. The top-left and bottom-right paths of the child path pair are the paths that extend the the top-left and bottom-right paths, respectively, of path pair $\mathcal{PP}$ with edge $\langle s, s' \rangle$ (Line 14).

After extracting a path pair $\mathcal{PP}$ from $Open$ that contains the goal state, PP-A* checks on Line 11 if there already exists a path pair in the solution set that (automatically contains the same state and) results in an ε-bounded path pair when merged with path pair $\mathcal{PP}$. If so, PP-A* removes that path pair from the solution set and then adds the merged path pair to the solution set (Lines 29-30). Otherwise, it adds path pair $\mathcal{PP}$ to the solution set (Line 32). Similarly, before generating a path pair $\mathcal{PP}$, PP-A* checks on Line 17 if there exists a path pair in $Open$ that contains the same state and results in an ε-bounded path pair when merged with path pair $\mathcal{PP}$. If so, PP-A* removes that path pair from $Open$ and then adds the merged path pair to $Open$. Otherwise, it adds path pair $\mathcal{PP}$ to $Open$.

When $Open$ becomes empty, PP-A* terminates and returns the bottom-right paths of all path pairs in the solution set as an ε-approximate Pareto-optimal solution set (Line 18).[2]

4 A*pex

In this section, we describe A*pex, our best-first multi-objective search algorithm that finds an ε-approximate Pareto-optimal solution set for a user-provided approximation factor ε.

In A*pex, as in PP-A*, nodes correspond to sets of paths with the same last state and similar costs. But A*pex improves on the representation of these sets, which 1) allows for larger and thus fewer sets of paths early in the search and thus results potentially in a search that is more efficient and produces smaller solution sets and 2) generalizes PP-A* from bi-objective to multi-objective search with any number of cost components. PP-A* includes only one path of a path pair in the solution set, namely the bottom-right path. The top-left path is not used for this purpose. The $\mathbf{g}$-value of a path pair is a vector whose first component is the smallest value of the first components of all paths in the set of paths that it represents and whose second component is the smallest value of the second components of all paths in this set. Again, the top-left path is not used for this purpose. **Contribution 1:** A*pex therefore represents a set of paths with a single representative path and a $\mathbf{g}$-value that is similar to the one of PP-A*. Having only one representative path instead of two provides flexibility. For example, the representative path can be chosen more freely than by PP-A*. **Contribution 2:** The representation of a set of paths can now easily be generalized from two to any number of cost components by extending the $\mathbf{g}$-value from a vector of size two to a vector of a size that equals the number of cost components. The ith component of the $\mathbf{g}$-value is the smallest value of the ith cost components of all paths in the set of paths that it represents.

Each node in $Open$ therefore corresponds to an ε-bounded *apex-path pair* $\mathcal{AP} = \langle \mathbf{A}, \pi \rangle$, where $\mathbf{A}$ is a vector of N cost components and π is a *representative path* with $\mathbf{A} \preceq \mathbf{g}(\pi)$. The apex-path pair contains state $s(\mathcal{AP}) = s(\pi)$ and $\mathbf{g}$-value (called apex) $\mathbf{g}(\mathcal{AP}) = \mathbf{A}$. The $\mathbf{g}$-value of an apex-path pair is the component-wise minimum of (and hence weakly dominates) the $\mathbf{g}$-values of all paths in the set of paths that it represents. **Contribution 3:** Similar to PP-A*, one could define an apex-path pair $\mathcal{AP} = \langle \mathbf{A}, \pi \rangle$ to be ε-bounded iff $\mathbf{g}(\pi) \preceq_{\boldsymbol{\varepsilon}} \mathbf{g}(\mathcal{AP})$, where the $\mathbf{g}$- and $\mathbf{f}$-values of paths are defined as for NAMOA* and NAMOA*dr. We generalize the preceding definition by defining the apex-path pair to be ε-bounded iff $\mathbf{f}(\pi) \preceq_{\boldsymbol{\varepsilon}} \mathbf{f}(\mathcal{AP})$, allowing for larger sets of paths to be represented by apex-path pairs early in the search.

Any two apex-path pairs that contain the same state can be *merged* into a single apex-path pair, where the apex of the merged apex-path pair is the component-wise minimum of the apexes of the two apex-path pairs and the representative path of the merged apex-path pair is either one of the two representative paths of the two apex-path pairs. See Figure 2 for a visualization of the two possible outcomes. Each of the two representative paths is considered a candidate for the merged apex-path pair if choosing it results in an ε-bounded merged apex-path pair. If there are no candidates, A*pex does not merge the apex-path pairs. Otherwise, which candidate it chooses does not affect its correctness but can affect its efficiency. We therefore consider these different methods:

- **Random (R) Method**: A*pex randomly chooses the representative path from the candidates.

[2] Returning the top-left paths of the path pairs in the solution set does not necessarily result in an ε-approximate Pareto-optimal solution set.

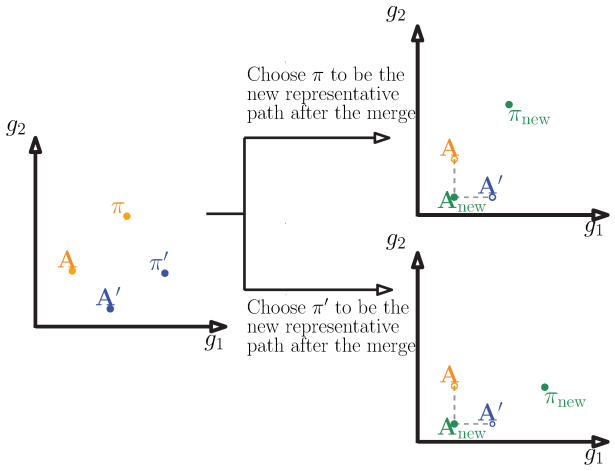

Figure 2: An example of merging apex-path pairs $\langle \mathbf{A}, \pi \rangle$ (orange) and $\langle \mathbf{A}', \pi' \rangle$ (blue) into Apex-path pair $\langle \mathbf{A}_{\text{new}}, \pi_{\text{new}} \rangle$ (green).

- **Lexicographically (L) Smallest Reverse g-Value Method**: A*pex chooses the representative path with the lexicographically smaller reverse g-value. If this path is not a candidate, A*pex does not merge the apex-path pairs. If there are only two cost components, this method is similar to how PP-A* merges path pairs, namely by picking the bottom-right path.
- **Greedy (G) Method**: A*pex chooses the candidate π with the larger slack

$$\min_{i=1.2...,N} \left\{ \frac{1 + \varepsilon_i - f_i(\pi)/f_i(\mathcal{AP})}{\varepsilon_i} \right\},$$

where $\mathcal{AP}$ is the resulting merged apex-path pair. The ith component of the $\mathbf{f}$-value of the representative path could be a factor of $1 + \varepsilon_i$ larger than the ith component of the $\mathbf{f}$-value of the apex-path pair but is only a factor of $f_i(\pi)/f_i(\mathcal{AP})$ larger. The smaller the difference of these two values, the better path π utilizes the leeway provided by the approximation factor. The difference, which can range from zero to ε_i, is divided by ε_i to normalize it and thus make the differences for different components comparable. Overall, the expression above indicates how much room is left to merge the merged apex-path pair with other apex-path pairs in the future, and maximizing it chooses the candidate that that leaves more room for future merges.

Algorithm 2 shows the pseudocode of A*pex. It starts with a single apex-path pair $\langle \mathbf{0}, [s_{\text{start}}] \rangle$ in *Open* (Line 1). At each iteration, A*pex extracts an apex-path pair from *Open* with the lexicographically smallest $\mathbf{f}$-value (Line 6). Similar to NAMOA* and NAMOA*dr, A*pex maintains a set $G_{\text{cl}}^T(s)$ for each state s that contains the undominated truncated $\mathbf{f}$-values of the expanded apex-path pairs that contain state s. This is necessary since the $f_2^{\min}(s)$ optimization of PP-A* only works for bi-objective search and affects dominance checking as follows. Both after extracting (that is, after Line 6) and before generating (that is, before

Algorithm 2: A*pex

Input : $P = \langle S, E, \mathbf{c}, s_{\text{start}}, G_{\text{cl}}^T() \rangle$
$\boldsymbol{h}$
$\boldsymbol{\varepsilon}$

1 $Open \leftarrow \{\langle \mathbf{0}, [s_{\text{start}}] \rangle\}$
2 solutions $\leftarrow \emptyset$
3 **foreach** $s \in S$ **do**
4 $\quad G_{\text{cl}}^T(s) \leftarrow \emptyset$
5 **while** $Open \neq \emptyset$ **do**
6 $\quad \mathcal{AP} = \langle \mathbf{A}, \pi \rangle \leftarrow Open$.extract_min()
7 $\quad$ **if** *is_dominated*($\mathcal{AP}$) **then**
8 $\quad\quad$ **continue**
9 $\quad G_{\text{cl}}^T(s(\mathcal{AP})).add(Tr(\mathbf{f}(\mathcal{AP}))$
10 $\quad$ **if** $s(\mathcal{AP}) = s_{goal}$ **then**
11 $\quad\quad$ insert($\mathcal{AP}$, solutions)
12 $\quad\quad$ **continue**
13 $\quad$ **for** $s' \in succ(s(\mathcal{AP}))$ **do**
14 $\quad\quad \mathcal{AP}' \leftarrow$
$\quad\quad\quad \langle \mathbf{A} + \mathbf{c}(\langle s(\mathcal{AP}), s' \rangle), \text{extend}(\pi, \langle s(\mathcal{AP}), s' \rangle) \rangle$
15 $\quad\quad$ **if** *is_dominated*($\mathcal{AP}'$) **then**
16 $\quad\quad\quad$ **continue**
17 $\quad\quad$ insert($\mathcal{AP}'$, *Open*)
18 **return** $\{\pi : \langle \mathbf{A}, \pi \rangle \in solutions\}$
19 **Function** *is_dominated*($\mathcal{AP} = \langle \mathbf{A}, \pi \rangle$)**:**
20 $\quad$ **if** $\exists \mathcal{AP}' = \langle \mathbf{A}', \pi' \rangle \in solutions$:
$\quad\quad Tr(\mathbf{f}(\pi')) \preceq_{\boldsymbol{\varepsilon}} Tr(\mathbf{f}(\mathcal{AP}))$ **then**
21 $\quad\quad$ remove $\langle \mathbf{A}', \pi' \rangle$ from solutions
22 $\quad\quad$ add $\langle comp\_wise\_minimum(\mathbf{f}(\mathcal{AP}), \mathbf{A}'), \pi' \rangle$ to solutions
23 $\quad\quad$ **return** *true*
24 $\quad$ **if** $\exists \mathbf{x} \in G_{cl}^T(s(\mathcal{AP})) : \mathbf{x} \preceq Tr(\mathbf{f}(\mathcal{AP}))$ **then**
25 $\quad\quad$ **return** *true*
26 $\quad$ **return** *false*
27 **Function** *insert*($\mathcal{AP}$, *list*)**:**
28 $\quad$ **for** $\mathcal{AP}' \in list$ **do**
29 $\quad\quad \mathcal{AP}_{\text{new}} \leftarrow merge(\mathcal{AP}, \mathcal{AP}')$
30 $\quad\quad$ **if** $\mathcal{AP}_{new}$ *is* $\boldsymbol{\varepsilon}$*-bounded* **then**
31 $\quad\quad\quad$ remove $\mathcal{AP}'$ from *list*
32 $\quad\quad\quad$ add $\mathcal{AP}_{\text{new}}$ to *list*
33 $\quad\quad\quad$ **return**
34 $\quad$ add $\mathcal{AP}$ to *list*
35 $\quad$ **return**

Line 17) an apex-path pair $\mathcal{AP}$ that contains state s, A*pex discards the apex-path pair

1. if there exists an expanded apex-path pair $\mathcal{AP}'$ that contains the goal state and the $\mathbf{f}$-value of whose representative path ε-dominates the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}$ (Line 20). A*pex also updates the apex of apex-path pair $\mathcal{AP}'$ to the component-wise minimum of the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}$ and the apex of $\mathcal{AP}'$ (Lines 21-22). This update guarantees that, if A*pex merges apex-path pair $\mathcal{AP}'$ with other apex-path pairs later on Line 29, the $\mathbf{f}$-value of the representative path of the resulting apex-path pair still ε-dominates the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}$ and thus also the $\mathbf{f}$-values of all paths in the set of

paths that apex-path pair $\mathcal{AP}$ represents. [3]

2. if there exists an expanded apex-path pair that contains state s and whose $\mathbf{f}$-value weakly dominates the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}$ (Line 24).

When A*pex expands an apex-path pair $\mathcal{AP}$ with state s, it generates a child apex-path pair for each successor state s' of state s. The apex of the child apex-path pair is the component-wise sum of the apex of apex-path pair $\mathcal{AP}$ and the cost of edge $\langle s, s' \rangle$, and the representative path of the child apex-path pair is the path the extends the representative path of apex-path pair $\mathcal{AP}$ with edge $\langle s, s' \rangle$ (Line 14).

Both after extracting an apex-path pair from $Open$ that contains the goal state and before generating an apex-path pair, A*pex attempts on Lines 11 and 17, respectively, to merge the apex-path pair with an apex-path pair in the solution set or $Open$, respectively, just like PP-A*.

When $Open$ becomes empty, A*pex terminates and returns the representative paths of all apex-path pairs in the solution set as an ε-approximate Pareto-optimal solution set (Line 18).

5 Theoretical Results

Lemma 1. *The sequence of extracted apex-path pairs has monotonically non-decreasing f_1-values.*

Proof. The proof is similar to the one of Lemma 3 in (Goldin and Salzman 2021). □

Lemma 2. *If there exists a truncated $\mathbf{f}$-value in $G_{cl}^T(s(\mathcal{AP}))$ that weakly dominates the truncated $\mathbf{f}$-value of some apex-path pair $\mathcal{AP}$ on Line 24, then there exists an expanded apex-path pair $\mathcal{AP}'$ that contains state $s(\mathcal{AP})$ and whose $\mathbf{f}$-value weakly dominates the one of apex-path pair $\mathcal{AP}$.*

Proof. Let apex-path pair $\mathcal{AP}'$ be the expanded apex-path pair that contains state $s(\mathcal{AP})$ and that Line 9 was executed with to add the truncated $\mathbf{f}$-value to $G_{\text{cl}}^T(s(\mathcal{AP}))$. It holds that $f_1(\mathcal{AP}') \leq f_1(\mathcal{AP})$ according to Lemma 1 and since the heuristic function is consistent. Thus, the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}'$ weakly dominates the one of apex-path pair $\mathcal{AP}$. □

Lemma 3. *If the apex of an apex-path pair weakly dominates a vector and the apex-path pair is merged with another apex-path pair, then the apex of the merged apex-path pair weakly dominates the vector as well.*

Proof. The apex of the merged apex-path pair is the component-wise minimum of the apexes of the two merged apex-path pairs. □

Lemma 4. *For any prefix $\pi_l = [s_1, s_2 \dots s_l]$ of any solution $\pi = [s_1(= s_{start}), s_2 \dots s_L(= s_{goal})]$ with $1 \leq l \leq L$, there exists, when A*pex terminates, (Case 1:) an expanded apex-path pair $\mathcal{AP}$ (that is, one that reaches Line 9) that contains state s_l and whose apex weakly dominates the $\mathbf{g}$-value of path π_l or (Case 2:) an apex-path pair $\mathcal{AP}$ in the solution set such that the $\mathbf{f}$-value of its representative path ε-dominates the $\mathbf{f}$-value of path π_l.*

Proof. The proof is by induction. The lemma holds for $l = 1$ and any solution since apex-path pair $\mathcal{AP} = \langle \mathbf{0}, [s_{\text{start}}] \rangle$ gets expanded and has the properties required for Case 1. Now assume that the lemma holds for some $l < L$ and any solution. We prove that it then also holds for $l + 1$ and this solution.

Assume that Case 1 holds for l and consider both the apex-path pair $\mathcal{AP}$ mentioned there and its potential child apex-path pair $\mathcal{AP}'$ created on Line 14 for $s' = s_{l+1}$. Apex-path pair $\mathcal{AP}'$ contains state s_{l+1}, and its apex weakly dominates the $\mathbf{g}$-value of path π_{l+1}, which implies that its $\mathbf{f}$-value weakly dominates the $\mathbf{f}$-value of path π_{l+1}. We distinguish several cases:

1. First, the condition on Line 20 holds for some apex-path pair in the solution set, namely, the truncated $\mathbf{f}$-value of the representative path of this apex-path pair ε-dominates the truncated $\mathbf{f}$-value of apex-path pair $\mathcal{AP}'$. A*pex replaces this apex-path pair with a new apex-path pair $\mathcal{AP}''$ in the solution set on Line 22. Apex-path pair $\mathcal{AP}''$ stays in the solution set but A*pex might merge it several (more) times with other apex-path pairs on Line 29 before it terminates. The apex of apex-path pair $\mathcal{AP}''$ weakly dominates the $\mathbf{f}$-value of path π_{l+1} (since this apex is the component-wise minimum of the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}'$ and another apex and hence weakly dominates the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}'$, which in turn weakly dominates the $\mathbf{f}$-value of path π_{l+1}) and merging it with other apex-path pairs does not change this property according to Lemma 3. Since the apex-path pair also remains ε-bounded (which is due to the conditions on Lines 20 and 30, Lemma 1, and since the heuristic function is consistent), the $\mathbf{f}$-value of its representative path always ε-dominates the $\mathbf{f}$-value of itself, which equals its apex. Put together, the $\mathbf{f}$-value of its representative path ε-dominates the $\mathbf{f}$-value of path π_{l+1}. Thus, the merged apex-path pair satisfies Case 2 for $l + 1$.
2. Second, the condition on Line 24 holds, namely, there exists a truncated $\mathbf{f}$-value in $G_{\text{cl}}^T(s(\mathcal{AP}'))$ that weakly dominates the truncated $\mathbf{f}$-value of apex-path pair $\mathcal{AP}'$.

[3] PP-A* does not need to perform such an update of the apex before Line 21 of Algorithm 1 for the following reason: Assume that $f_2^{\min}(s_{\text{goal}}) \leq (1 + \varepsilon_2) f_2(\mathcal{PP})$ (Equation 1) on Line 20 for some path pair $\mathcal{PP} = \langle \pi^{\text{tl}}, \pi^{\text{br}} \rangle$. Assume further that path pair $\mathcal{PP}' = \langle \pi^{\text{tl}'}, \pi^{\text{br}'} \rangle$ resulted in the value of $f_2^{\min}(s_{\text{goal}})$ via the assignment on Line 9 and is later merged with another path pair. Then, the $\mathbf{f}$-value of the bottom-right path of the resulting path pair $\mathcal{PP}'' = \langle \pi^{\text{tl}''}, \pi^{\text{br}''} \rangle$ still ε-dominates the $\mathbf{f}$-value of path pair $\mathcal{PP}$ since the $\mathbf{f}$-values of solutions are equal to their $\mathbf{g}$-values and 1) $f_1(\pi^{\text{br}''}) \leq (1+\epsilon_1) f_1(\pi^{\text{tl}''}) \leq (1+\epsilon_1) f_1(\pi^{\text{tl}'}) \leq (1+\epsilon_1) f_1(\pi^{\text{tl}})$ (since, in order of the $\leq$ relationships, $\mathcal{PP}''$ is ε-bounded, the first component of the $\mathbf{g}$-value of the path pair resulting from merging is no larger than those of the merged path pairs, and the sequence of the first components of the $\mathbf{g}$-values of expanded path pairs is monotonically non-decreasing) and 2) $f_2(\pi^{\text{br}''}) \leq f_2(\pi^{\text{br}'}) \leq (1 + \epsilon_2) f_2(\pi^{\text{br}})$ (since, in order of the $\leq$ relationships, the second component of the $\mathbf{g}$-value of the path pair resulting from merging is no larger than those of the merged path pairs and Equation 1 holds).

Then, an expanded apex-path pair $\mathcal{AP}''$ exists according to Lemma 2 that contains state s_{l+1} and whose $\mathbf{f}$-value weakly dominates the $\mathbf{f}$-value of apex-path pair $\mathcal{AP}'$. Thus, its apex weakly dominates the apex of apex-path pair $\mathcal{AP}'$. Thus, apex-path pair $\mathcal{AP}''$ satisfies Case 1 for $l+1$ since the apex of apex-path pair $\mathcal{AP}'$ in turn weakly dominates the $\mathbf{g}$-value of path π_{l+1}.

3. Otherwise, A*pex executes Line 17 for apex-path pair $\mathcal{AP}'$, where the apex-path pair is inserted into $Open$, perhaps after having been merged with another apex-path pair on Line 29. A*pex might merge it several (more) times with other apex-path pairs on Line 29 before finally extracting it. Its apex weakly dominates the $\mathbf{g}$-value of path π_{l+1} and merging it with other apex-path pairs does not change this property according to Lemma 3. Thus, if this apex-path pair is expanded, it satisfies Case 1 for $l+1$. If it is extracted but not expanded, the condition on Line 20 or Line 24 holds, and thus, as we have already proved, Case 1 or Case 2 holds.

Assume that Case 2 holds for l and consider the apex-path pair mentioned there. The $\mathbf{f}$-value of the representative path of this apex-path pair ε-dominates the $\mathbf{f}$-value of path π_l. Since the heuristic function is consistent, the $\mathbf{f}$-value of path π_l in turn weakly dominates the $\mathbf{f}$-value of path π_{l+1}. Thus, this apex-path pair satisfies Case 2 for $l+1$. □

The following theorem shows that A*pex determines an ε-approximate Pareto-optimal solution set.

Theorem 1. *For any solution π, there exists, when A*pex terminates, an apex-path pair in the solution set whose representative path ε-dominates π.*

Proof. Lemma 4 holds for prefix $\pi_L = \pi$ of any solution π. In case its Case 2 holds, the theorem holds by definition for path π since the $\mathbf{f}$-values of solutions (including those of the representative path and path π) are equal to their costs. In case its Case 1 holds, consider the apex-path pair mentioned there. This apex-path pair contains the goal state, and A*pex thus executed Line 11 for it, where the apex-path pair was inserted into the solution set, perhaps after having been merged with another apex-path pair on Line 29. The apex-path pair stays in the solution set but A*pex might merge it several (more) times with other apex-path pairs on Line 29 before it terminates. The apex of the apex-path pair weakly dominates the $\mathbf{g}$-value of path π according to Lemma 4 and merging it with other apex-path pairs does not change this property according to Lemma 3. Since the apex-path pair also remains ε-bounded (which is due to the conditions on Lines 20 and 30, Lemma 1, and since the heuristic function is consistent), the $\mathbf{f}$-value of its representative path always ε-dominates the $\mathbf{f}$-value of itself, which equals its apex. Put together, the $\mathbf{f}$-value of its representative path ε-dominates the $\mathbf{g}$-value of path π. Thus, the theorem holds by definition for path π since the $\mathbf{g}$- and $\mathbf{f}$-values of solutions (including those of the representative path and path π) are equal to their costs. □

BAY				
321,270 states, 794,830 edges				
ε	PP-A*	A*pex-R	A*pex-L	A*pex-G
0.0001	3.9 (1.00)	3.3 (1.00)	3.3 (1.00)	3.2 (1.00)
0.001	3.1 (1.00)	1.7 (1.00)	1.7 (1.00)	1.5 (1.00)
0.005	1.9 (1.00)	1.1 (1.00)	1.0 (1.00)	1.0 (1.00)
0.01	1.4 (1.00)	1.0 (1.00)	0.9 (1.00)	0.9 (1.00)
0.025	1.0 (1.00)	0.9 (1.00)	0.7 (1.00)	0.8 (1.00)
0.05	0.6 (1.00)	0.7 (1.00)	0.5 (1.00)	0.6 (1.00)
0.1	0.4 (1.00)	0.5 (1.00)	0.4 (1.00)	0.4 (1.00)
0.5	0.1 (1.00)	0.1 (1.00)	0.1 (1.00)	0.1 (1.00)
1	0.0 (1.00)	0.0 (1.00)	0.0 (1.00)	0.0 (1.00)

FLA				
1,070,376 states, 2,712,798 edges				
ε	PP-A*	A*pex-R	A*pex-L	A*pex-G
0.0001	26.5 (0.92)	15.7 (1.00)	17.2 (1.00)	15.3 (1.00)
0.001	30.9 (1.00)	10.3 (1.00)	9.8 (1.00)	8.4 (1.00)
0.005	14.3 (1.00)	5.4 (1.00)	4.1 (1.00)	4.4 (1.00)
0.01	9.1 (1.00)	4.7 (1.00)	3.6 (1.00)	3.7 (1.00)
0.025	4.3 (1.00)	4.0 (1.00)	2.6 (1.00)	2.7 (1.00)
0.05	2.4 (1.00)	3.1 (1.00)	1.9 (1.00)	2.0 (1.00)
0.1	1.3 (1.00)	1.7 (1.00)	1.2 (1.00)	1.2 (1.00)
0.5	0.3 (1.00)	0.2 (1.00)	0.2 (1.00)	0.2 (1.00)
1	0.1 (1.00)	0.1 (1.00)	0.1 (1.00)	0.1 (1.00)

NE				
1,524,453 states, 3,897,636 edges				
ε	PP-A*	A*pex-R	A*pex-L	A*pex-G
0.0001	53.8 (0.72)	37.5 (0.80)	40.1 (0.80)	37.2 (0.80)
0.001	52.0 (0.80)	21.2 (0.88)	22.7 (0.88)	18.2 (0.96)
0.005	46.1 (0.92)	17.6 (1.00)	16.2 (0.96)	14.0 (1.00)
0.01	34.4 (0.96)	15.8 (1.00)	12.9 (1.00)	13.3 (1.00)
0.025	25.1 (1.00)	15.9 (1.00)	11.9 (1.00)	12.6 (1.00)
0.05	14.8 (1.00)	14.5 (1.00)	9.8 (1.00)	10.2 (1.00)
0.1	9.1 (1.00)	11.1 (1.00)	7.6 (1.00)	7.8 (1.00)
0.5	1.2 (1.00)	1.3 (1.00)	1.3 (1.00)	1.3 (1.00)
1	0.2 (1.00)	0.2 (1.00)	0.2 (1.00)	0.2 (1.00)

Table 1: Average runtimes (in seconds) and, in parentheses, success rates on bi-objective road map instances for different approximation factors ε.

6 Evaluation

In this section, we compare the efficiency of PP-A*, NAMOA*dr-ε, A*pex with the Random Method (A*pex-R), A*pex with the Lexicographically Smallest Reverse $\mathbf{g}$-Value Method (A*pex-L), and A*pex with the Greedy Method (A*pex-G) on the road maps BAY, FLA, and NE of the 9th DIMACS Implementation Challenge: Shortest Path.[4] We ran all experiments on t2.large AWS EC2 instances with 8GB of memory and a runtime limit of five minutes per instance. We implemented all algorithms in C++, using a common code base as much as possible.[5]

We generated 25 bi- and tri-objective road map instances with randomly selected start and goal states for each road map. The first two components c_1 and c_2 of the cost function represent travel distances and times, respectively, which

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

[5] https://github.com/HanZhang39/A-pex.

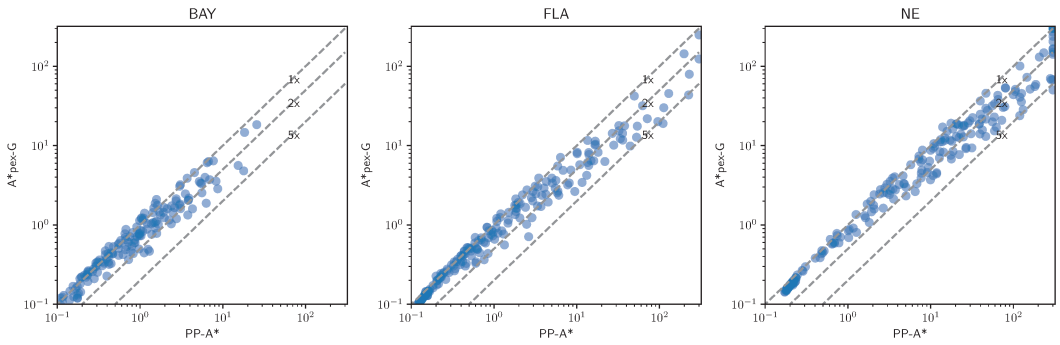

Figure 3: Runtimes of PP-A* and A*pex-G on bi-objective road map instances.

BAY				
ε	PP-A*	A*pex-R	A*pex-L	A*pex-G
0.001	56.0	29.8	27.8	28.0
0.01	15.5	7.6	5.4	5.3
0.1	3.2	1.1	1.1	1.1

FLA				
ε	PP-A*	A*pex-R	A*pex-L	A*pex-G
0.001	81.4	68.4	40.5	53.3
0.01	13.0	8.6	3.9	4.1
0.1	2.5	1.1	1.1	1.1

NE				
ε	PP-A*	A*pex-R	A*pex-L	A*pex-G
0.001	125.0	98.1	71.0	90.7
0.01	25.2	23.6	9.4	10.7
0.1	3.9	1.2	1.1	1.1

Table 2: Average sizes of solution sets on bi-objective road map instances for different approximation factors ε.

are both available from the DIMACS data set. We randomly generated the third component, since it is not available from the DIMACS data set, as $c_3(e) = x(e)(c_1(e) + c_2(e))$ for all edges $e \in E$, where $x(e)$ is chosen uniformly at random from the interval $[0.3, 0.4]$ for each edge. Thus, the third component is positively but not linearly correlated with the first two components. We used the vector of the minimum cost from state s to the goal state for each cost objective, computed with Dijkstra's algorithm, as $\mathbf{h}$-value of state s, as is common in the multi-objective search literature since the computation of all $\mathbf{h}$-values (which took at most around 3 seconds per instance in our experiments) is only a small fraction of the runtime. We reported the runtime without this computation since all search algorithms compute the same $\mathbf{h}$-values. We used approximation factors $\boldsymbol{\varepsilon} = [\varepsilon, \varepsilon \dots \varepsilon]$, denoted in the following simply as ε.

6.1 Bi-Objective Road Map Instances

Table 1 shows the average runtimes (in seconds) of PP-A*, A*pex-R, A*pex-L, and A*pex-G over all bi-objective road map instances that were solved by all four algorithms within the runtime limit and, in parentheses, the success rates (defined as the percentages of instances solved within the runtime limit). We omit the results for NAMOA*dr-ε because PP-A* significantly outperformed BOA*-ε on the instances in (Goldin and Salzman 2021), and BOA*-ε improves the efficiency of the dominance checks of NAMOA*dr-ε. The success rates of A*pex-R and A*pex-G were at least as large as the ones of A*pex-L for all approximation factors and road maps, and the success rates of A*pex-L were at least as large as the ones of PP-A* for all approximation factors and road maps and larger than the ones of PP-A* for $\varepsilon \leq 0.01$ on road map NE. The runtimes of A*pex-R, A*pex-L, and A*pex-G were smaller than the ones of PP-A* for many approximation factors and road maps. The runtimes of A*pex-R were similar to the ones of A*pex-G for small (≤ 0.005) or large (≥ 0.1) approximation factors but larger than the ones of A*pex-L and A*pex-G for approximation factors in between. The runtimes of A*pex-L were larger than the ones of A*pex-G for small (≤ 0.001) approximation factors but similar to the ones of A*pex-G otherwise.

Figure 3 shows the individual runtimes (in seconds) of PP-A* and A*pex-G for all approximation factors and road map instances. The runtimes of PP-A* and A*pex-G were similar on easy instances (where the runtimes were small). However, the runtimes of A*pex-G were much smaller than the ones of PP-A* on hard instances (where the runtimes were large), and A*pex-G solved a few instances that PP-A* could not solve within the runtime limit (shown on the right boundary in the plots). A comparison of PP-A* and A*pex-R and one of PP-A* and A*pex-L yielded similar results.

Table 2 shows the average sizes of the solution sets of PP-A*, A*pex-R, A*pex-L, and A*pex-G on all bi-objective road map instances that were solved by all four algorithms within the runtime limit. We show only the results for a representative set of approximation factors due to the space limit. All three A*pex variants found solution sets of smaller average sizes than PP-A* for all approximation factors and road maps. The solution sets of A*pex-L and A*pex-G were of similar average sizes and no larger than the ones of A*pex-R (but often smaller).

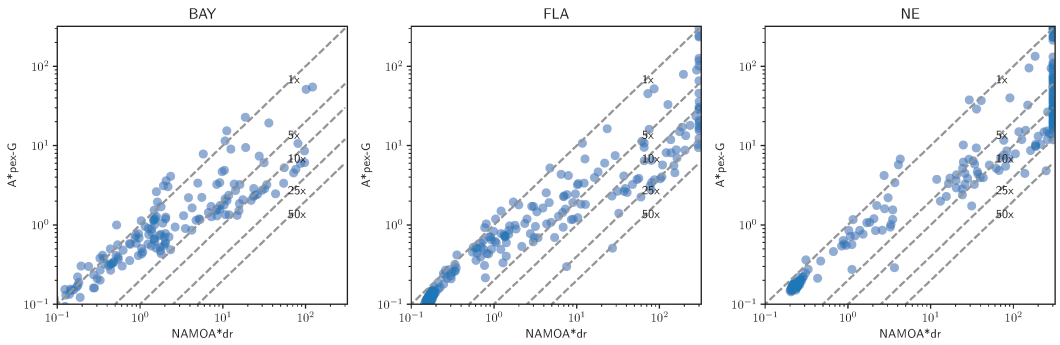

Figure 4: Runtimes of NAMOA*dr-ε and A*pex-G on tri-objective road map instances.

BAY				
321,270 states, 794,830 edges				
ε	NAMOA*dr-ε	A*pex-R	A*pex-L	A*pex-G
.0001	15.2 (1.00)	9.9 (1.00)	13.3 (1.00)	9.3 (1.00)
0.001	12.5 (1.00)	4.8 (1.00)	3.0 (1.00)	2.4 (1.00)
0.005	12.2 (1.00)	1.7 (1.00)	1.5 (1.00)	1.5 (1.00)
0.01	11.1 (1.00)	2.4 (1.00)	1.3 (1.00)	1.4 (1.00)
0.025	8.0 (1.00)	1.4 (1.00)	1.1 (1.00)	1.1 (1.00)
0.05	5.9 (1.00)	1.1 (1.00)	0.8 (1.00)	0.8 (1.00)
0.1	2.9 (1.00)	1.0 (1.00)	0.5 (1.00)	0.5 (1.00)
0.5	0.1 (1.00)	0.1 (1.00)	0.1 (1.00)	0.1 (1.00)
1	0.1 (1.00)	0.0 (1.00)	0.0 (1.00)	0.0 (1.00)

FLA				
1,070,376 states, 2,712,798 edges				
ε	NAMOA*dr-ε	A*pex-R	A*pex-L	A*pex-G
0.0001	93.9 (0.80)	59.3 (0.92)	69.7 (0.88)	54.5 (0.96)
0.001	85.4 (0.88)	19.1 (1.00)	15.3 (1.00)	12.8 (1.00)
0.005	72.7 (0.88)	11.0 (1.00)	6.3 (1.00)	6.5 (1.00)
0.01	68.1 (0.88)	9.3 (1.00)	5.3 (1.00)	5.6 (1.00)
0.025	50.9 (0.92)	5.9 (1.00)	3.9 (1.00)	3.9 (1.00)
0.05	42.0 (0.92)	4.3 (1.00)	2.7 (1.00)	2.9 (1.00)
0.1	27.3 (1.00)	2.7 (1.00)	1.8 (1.00)	1.8 (1.00)
0.5	1.6 (1.00)	0.3 (1.00)	0.3 (1.00)	0.3 (1.00)
1	0.2 (1.00)	0.1 (1.00)	0.1 (1.00)	0.1 (1.00)

NE				
1,524,453 states, 3,897,636 edges				
ε	NAMOA*dr-ε	A*pex-R	A*pex-L	A*pex-G
0.0001	177.8 (0.52)	165.8 (0.52)	170.5 (0.52)	158.4 (0.64)
0.001	172.6 (0.52)	99.3 (0.80)	97.8 (0.80)	78.1 (0.88)
0.005	172.2 (0.52)	55.0 (0.96)	37.3 (0.96)	32.5 (1.00)
0.01	169.9 (0.52)	42.9 (0.96)	23.9 (1.00)	23.8 (1.00)
0.025	158.9 (0.60)	24.1 (1.00)	17.5 (1.00)	18.7 (1.00)
0.05	147.8 (0.60)	23.1 (1.00)	14.6 (1.00)	15.3 (1.00)
0.1	128.3 (0.72)	22.9 (1.00)	11.5 (1.00)	11.7 (1.00)
0.5	26.5 (0.96)	1.9 (1.00)	1.8 (1.00)	1.9 (1.00)
1	0.3 (1.00)	0.2 (1.00)	0.2 (1.00)	0.2 (1.00)

Table 3: Average runtimes (in seconds) and, in parentheses, success rates on tri-objective road map instances for different approximation factors ε.

6.2 Tri-Objective Road Map Instances

Table 3 shows the average runtimes (in seconds) of NAMOA*dr-ε, A*pex-R, A*pex-L, and A*pex-G over all

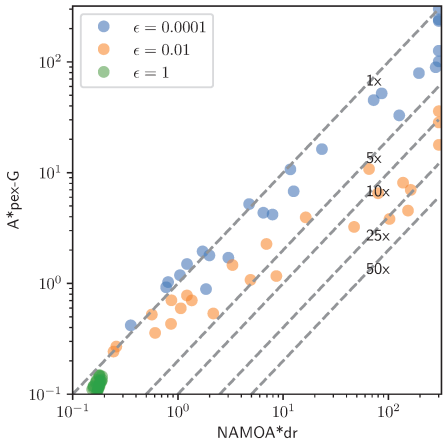

Figure 5: Runtimes of NAMOA*dr-ε and A*pex-G on tri-objective road map instances of road map FLA with approximation factors $\varepsilon = 0.0001$, 0.01, and 1.

BAY				
ε	NAMOA*dr-ε	A*pex-R	A*pex-L	A*pex-G
0.001	69.8	38.4	32.2	31.7
0.01	16.2	7.0	5.6	5.4
0.1	3.2	1.2	1.1	1.1

FLA				
ε	NAMOA*dr-ε	A*pex-R	A*pex-L	A*pex-G
0.001	74.8	56.9	29.7	31.9
0.01	11.6	7.3	3.2	3.5
0.1	2.4	1.0	1.1	1.1

NE				
ε	NAMOA*dr-ε	A*pex-R	A*pex-L	A*pex-G
0.001	118.4	92.6	69.8	66.0
0.01	21.9	14.2	7.8	8.7
0.1	3.6	1.2	1.1	1.1

Table 4: Average sizes of solution sets on tri-objective road map instances for different approximation factors ε.

tri-objective road map instances, similar to Table 1. The results for NAMOA*dr-ε on tri-objective instances were similar to the ones for PP-A* on bi-objective instances, with the following difference: The runtimes of A*pex-L and A*pex-G were again similar to each other, now except for approximation factor $\varepsilon = 0.0001$, where the speed-up of A*pex-G over A*pex-L was up to a factor of about 1.4. The runtimes of A*pex-R were much larger than the ones of A*pex-G and A*pex-L for more values of the approximation factor (from 0.001 to 0.1) on maps FLA and NE.

Figure 4 shows the individual runtimes (in seconds) of NAMOA*dr-ε and A*pex-G for all approximation factors and road map instances, similar to Figure 3. Again, the results for NAMOA*dr-ε on tri-objective instances were similar to the ones for PP-A* on bi-objective instances, with the following difference: The speed-up of A*pex-G over NAMOA*dr-ε was up to a factor of 50, and A*pex-G solved several instances that NAMOA*dr-ε could not solve within the runtime limit. Figure 5 is similar to Figure 4 for road map FLA, but the results are now color-coded for a representative subset of approximation factors. Colors indicate the different approximation factors. The speed-ups of A*pex-G over NAMOA*dr-ε for both the small approximation factor ($\varepsilon = 0.0001$) and the large one ($\varepsilon = 1$) were much smaller than the speed-ups for the one in between ($\varepsilon = 0.01$) because A*pex-G did not have many opportunities to merge nodes in the former case. The speed-up for $\varepsilon = 0.01$ was more than a factor of 25 for two instances,

Table 4 shows the average sizes of the solution sets of NAMOA*dr-ε, A*pex-R, A*pex-L, and A*pex-G on all tri-objective road map instances that were solved by all four algorithms within the runtime limit, similar to Table 2. The results for NAMOA*dr-ε on tri-objective instances were similar to the ones for PP-A* on bi-objective instances.

7 Conclusions

In this paper, we presented A*pex, a multi-objective search algorithm that finds an ε-approximate Pareto-optimal solution set for a user-provided approximation factor ε. It builds on PP-A* but (1) makes PP-A* more efficient for bi-objective search and (2) generalizes it to multi-objective search for any number of cost components. We first analyzed the correctness of A*pex and then experimentally demonstrated its efficiency advantage over PP-A* for bi-objective search and a baseline algorithm derived from NAMOA*dr for tri-objective search. It is future work to extend A*pex to bi-directional and anytime search.

8 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. The research was also supported by the United States-Israel Binational Science Foundation (BSF) under grant number 2021643 and Centro Nacional de Inteligencia Artificial CENIA, FB210017, 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 any government.

References

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.

Hernandez, C. U.; Yeohz, W.; Baier, J. A.; Zhang, H.; Suazoy, L.; and Koenig, S. 2020. A Simple and Fast Bi-Objective Search Algorithm. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 143–151.

Mandow, L.; and De La Cruz, J. L. P. 2010. Multiobjective A* Search with Consistent Heuristics. *Journal of the ACM*, 57(5): 1–25.

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.

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.