Artificial Intelligence 339 (2025) 104260

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint

EMOA*: A framework for search-based multi-objective path planning

Zhongqiang Ren [a,d], Carlos Hernández [b,c,*], Maxim Likhachev [d], Ariel Felner [e], Sven Koenig [f], Oren Salzman [g], Sivakumar Rathinam [h], Howie Choset [d]

[a] UM-SJTU Joint Institute, Shanghai Jiao Tong University, China
[b] Facultad de Ingeniería, Arquietectura y Diseño, Universidad San Sebastián, Bellavista 7, Santiago, Chile
[c] Centro Científico y Tecnológico de Excelencia Ciencia & Vida, Chile
[d] Robotics Institute, Carnegie Mellon University, USA
[e] Department of Information System Engineering, at Ben-Gurion University, Israel
[f] Department of Computer Science, University of Southern California, USA
[g] Department of Computer Science, Technion–Israel Institute of Technology, Israel
[h] Department of Mechanical Engineering, Texas A&M University, USA

ARTICLE INFO

Keywords:
Heuristic search
Multi-objective optimization
Path planning

ABSTRACT

In the Multi-Objective Shortest Path Problem (MO-SPP), one has to find paths on a graph that simultaneously minimize multiple objectives. It is not guaranteed that there exists a path that minimizes all objectives, and the problem thus aims to find the set of Pareto-optimal paths from the start to the goal vertex. A variety of multi-objective A*-based search approaches have been developed for this purpose. Typically, these approaches maintain a front set at each vertex during the search process to keep track of the Pareto-optimal paths that reach that vertex. Maintaining these front sets becomes burdensome and often slows down the search when there are many Pareto-optimal paths. In this article, we first introduce a framework for MO-SPP with the key procedures related to the front sets abstracted and highlighted, which provides a novel perspective for understanding the existing multi-objective A*-based search algorithms. Within this framework, we develop two different, yet closely related approaches to maintain these front sets efficiently during the search. We show that our approaches can find all cost-unique Pareto-optimal paths, and analyze their runtime complexity. We implement the approaches and compare them against baselines using instances with three, four and five objectives. Our experimental results show that our approaches run up to an order of magnitude faster than the baselines.

1. Introduction

Given a graph with non-negative scalar edge costs, the Shortest Path Problem (SPP) calls for computing a minimum cost path from a given start vertex to a given destination vertex in the graph. In this article, we consider the Multi-Objective Shortest Path Problem (MO-SPP) [1–4], which generalizes SPP by associating each edge in the graph with a non-negative *cost vector* of constant length, where

* Corresponding author at: Facultad de Ingeniería, Arquietectura y Diseño, Universidad San Sebastián, Bellavista 7, Santiago, Chile.
E-mail addresses: zhongqiang.ren@sjtu.edu.cn (Z. Ren), carlos.hernandez@uss.cl (C. Hernández), mlikhach@andrew.cmu.edu (M. Likhachev), felner@bgu.ac.il (A. Felner), skoenig@usc.edu (S. Koenig), osalzman@cs.technion.ac.il (O. Salzman), srathinam@tamu.edu (S. Rathinam), choset@andrew.cmu.edu (H. Choset).

https://doi.org/10.1016/j.artint.2024.104260
Received 19 June 2023; Received in revised form 11 November 2024; Accepted 21 November 2024

Available online 2 December 2024

each component of the vector corresponds to an objective to be minimized. MO-SPP arises in many applications, including hazardous material transportation [5], robot design [6], robot inspection planning [7,8] and airport departure runway scheduling [9].

For example, the hazardous material transportation [5] is a path planning problem from a starting location to a destination in an urban area. The problem calls for computing the shortest path to transport the material while accounting for vulnerable centres such as schools, hospitals, etc. This requires balancing travel distance and risk of exposure (see, e.g., [10]). The problem has been efficiently solved using heuristic-search approaches for the bi-objective case [11].

For MO-SPP, it is not guaranteed that there exists a path that simultaneously optimizes all objectives. MO-SPP thus seeks to find a Pareto-optimal set of paths, whose cost vectors form the Pareto-optimal front. Here, a path is Pareto-optimal (or, synonymously, non-dominated) if there is no other path that can decrease one cost without increasing at least one of the other costs. Unfortunately, computing the Pareto-optimal front is challenging [12] as its cardinality may be exponential in the number of vertices [13–15].

To solve MO-SPP, several multi-objective A* (MOA*)-like planners [2,3,11,16–18] have been developed to compute the exact or an approximate Pareto-optimal front. In MO-SPP, there are typically multiple non-dominated paths from the start vertex to any other vertex in the graph, and MOA*-like planners store, select and expand these non-dominated paths at each vertex during the search. When a new path π to some vertex v is found, π needs to be compared with all previously found non-dominated paths to v to check for dominance, namely, to verify whether π is dominated by any other existing paths that reach v. These dominance checks are computationally expensive, especially when there are many non-dominated paths at a vertex, as it requires many cost vector comparisons [19].

To find the Pareto-optimal front efficiently, techniques have recently been developed to expedite these dominance checks for MOA*-like planners [11,19]. Among them, Bi-Objective A* (BOA*) [11] achieves around an order of magnitude speed-up over previous state-of-the-art MOA*-like search algorithms. Recently, BOA* has been improved further [16,17]. However, BOA* and its improved versions can handle only two objectives. We thus develop fast general dominance-checking methods that can handle an arbitrary number of objectives. To this end, we propose a search framework called Enhanced Multi-Objective A* (EMOA*), which abstracts and highlights the key procedures related to these expensive dominance checks during the MOA* search. This framework provides a novel perspective to understanding various existing MOA*-like search algorithms while highlighting the computational bottleneck bypassed in this article. The specific enhancement in EMOA* against the existing MOA* search is the identification of key sub-problems during the search process, and the use of fast data structure and algorithms to solve these key sub-problems during the search. Furthermore, we show that BOA* is a specific instantiation of EMOA* when there are only two objectives.

Within the EMOA* framework, we further develop two different, yet closely-related algorithms for fast dominance checking, by leveraging the existing ideas and techniques in the literature [20–23]. Both algorithms can handle an arbitrary number of objectives. Specifically, we first develop a new method that uses a balanced binary search tree (BBST) to store the non-dominated paths at each vertex. The key ideas are: (i) the BBST can be *incrementally* constructed during the MOA* search and is computationally efficient to maintain; (ii) the BBST is organized using the lexicographic order between cost vectors, which *guides* the dominance checks and expedites the computation; and (iii) our BBST-based method is compatible with existing approaches for fast dominance checking, which allows us to use both the existing and our newly developed techniques together to speed up the computation of the Pareto-optimal front.

As an alternative to this BBST-based algorithm, we also propose a second algorithm based on lexicographically sorted lists and Binary Search (BS), which further improves the computational efficiency of the BBST-based approach by reducing the frequent rotation operations needed to re-balance the search tree.

We show that both the BBST-based and BS-based algorithms are guaranteed to find the exact Pareto-optimal front for MO-SPP. We analyze the runtime complexity of the proposed methods. To verify our EMOA* framework, we implement several algorithms that follow our framework by using different data structures and approaches for dominance checking. We compare these implementations on instances with three, four and five objectives. Our experimental results show that our methods run up to an order of magnitude faster than an existing state-of-the-art method [19].

Preliminary versions of this work appeared in [24] and [25] which presented two instantiations of EMOA*. This article differs from the aforementioned work by introducing the general EMOA* framework we present, including a comprehensive description and discussion of EMOA* as well as a detailed analysis of the two algorithms that appeared in [24] and [25], including proofs of their solution quality guarantees and runtime complexities. Finally, we present new experimental results of both algorithms in various test settings.

1.1. Related work

Research on MO-SPP and its variants has a long history [1–3] and remains an active research topic [11,16,17,26,27]. Algorithms that solve MO-SPP range from exact methods [2,3,11] to approximation methods [16,28–31], trading off solution quality for computational efficiency. Among the exact methods, Multi-Objective A* [2] is one of the first approaches to extend the well-known A* algorithm [32] to address multiple objectives, and was later revised and expedited by *A New Approach to Multi-Objective A** (NAMOA*) [33]. NAMOA* was further improved and led to NAMOA*dr [19], where they use a technique based on "dimensionality reduction", which can reduce the length of the vectors by one when running dominance checks during the A*-like search. All these three algorithms [2,19,33] can address an arbitrary number of objectives. The recent BOA* [11] further expedites NAMOA*dr by introducing the idea of lazy dominance checks which allows to perform dominance checks in constant-time. However, BOA* is limited to bi-objective problems only. BOA* and NAMOA*dr, as state-of-the-art algorithms for bi-objective problems and general

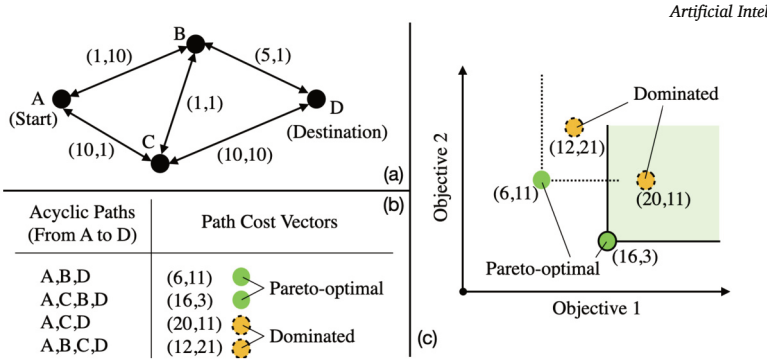

Fig. 1. An illustration of dominance. (a) An MO-SPP problem with two objectives ($M = 2$). (b) Visualization of all solutions (without loops) and their corresponding cost vectors, where the green dots are the Pareto-optimal cost vectors and the yellow dots are the dominated ones. (c) Visualization of the solution cost vectors. The x-axis corresponds to objective 1 and the y-axis corresponds to objective 2. The green region in (c) visualizes the set of vectors that are dominated by $(16, 3)$. Note that $(16, 3)$ dominates $(20, 11)$, and $(6, 11)$ dominates both $(12, 21)$ and $(20, 11)$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

multi-objective problems respectively, are two closely related algorithms to this article, and will be revisited with more details in Sec. 3.

As the aforementioned algorithms extend A*, they can make use of a heuristic function to expedite the search. In the single-objective setting, the search needs to consider only one minimum cost path from any vertex to the destination, and the heuristic function maps each vertex to a single-value which estimates the cost-to-go to reach the destination from that vertex. In contrast to the single-objective setting, from every vertex there may be multiple paths that belong to the Pareto-optimal front in the presence of multiple objectives. Thus, a heuristic function may map each vertex to a set of values, i.e., a set of cost vectors that estimate the cost-to-go to reach the destination from that vertex. Such a heuristic is called a *multi-valued heuristic* [3]. While recent papers explored the use of multi-valued heuristics (see, e.g., [34,35]), the lion's share of studies on MO-SPP use a *single-valued heuristic* where we store only one cost vector per vertex. This paper considers only single-valued heuristics.

Additionally, given a set of vectors, to efficiently compute the subset of vectors that are mutually non-dominated, a variety of approaches have been developed. For example, given a set of n vectors, Kung's method [20] has a worst-case runtime complexity $O(n \log n)$ when the vectors are of length two or three, and $O(n \log^{m-2} n)$ when the vectors are of length $m, m > 3$. Other approaches (e.g., [21,22]) run in $O(mn)$ time on average. Additionally, tree-based data structure has also been leveraged to speed up dominance comparisons [23].

2. Problem description

Let $G = (V, E, \vec{c})$ denote a finite directed graph with vertex set V and edge set E, where each edge $e = (u, v) \in E$ is associated with a non-negative cost vector $\vec{c}(e) = \vec{c}(u, v) \in (\mathbb{R}^+)^M$ with M being a positive integer and $\mathbb{R}^+$ being the set of non-negative real numbers. Let $\pi(v_1, v_\ell)$ denote a path connecting $v_1, v_\ell \in V$ via a sequence of vertices $(v_1, v_2, \ldots, v_\ell)$ in G, where v_n and v_{n+1} are connected by an edge $(v_n, v_{n+1}) \in E$, for $n = 1, 2, \ldots, \ell - 1$. Let $\vec{g}(\pi(v_1, v_\ell))$ denote the cost vector corresponding to the path $\pi(v_1, v_\ell)$, which is the sum of the cost vectors of all edges present in the path, i.e., $\vec{g}(\pi(v_1, v_\ell)) = \Sigma_{n=1}^{\ell-1} \vec{c}(v_n, v_{n+1})$. We use c_k with a subscript k to denote the k-th component of the vector $\vec{c}$. To compare any two paths, we compare the cost vectors associated with them using the dominance relation [14]. Intuitively, given two vectors $\vec{a}$ and $\vec{b}$, we say $\vec{a}$ dominates $\vec{b}$ if (i) every value of $\vec{a}$ is lower than or equal to the corresponding value of $\vec{b}$ and (ii) at least one value of $\vec{a}$ is strictly lower than the corresponding value of $\vec{b}$. Formally, we have the following definition.

Definition 1 *(Dominance).* Given two vectors $\vec{a}$ and $\vec{b}$ of length K $(K \geq 2)$,[1] $\vec{a}$ dominates $\vec{b}$ (denoted as $\vec{a} \prec \vec{b}$) [2] if and only if $\forall k \in \{1, 2, \ldots, K\}$, $a_k \leq b_k$, and $\exists k \in \{1, 2, \ldots, K\}$ such that $a_k < b_k$.

Let v_o, v_d denote the start and destination vertices, respectively. A path from v_o to v_d is also called a *solution*. One solution π_1 dominates another solution π_2 if the cost vector $\vec{g}(\pi_1)$ dominates $\vec{g}(\pi_2)$. The set of all non-dominated solutions is called the *Pareto-optimal set*. A subset of the Pareto-optimal set, where any two solutions in this subset do not have the same cost vector is called a *cost-unique* Pareto-optimal set. Fig. 1 provides an illustration of these concepts.

Definition 2 *(MO-SPP).* Given a graph $G = (V, E, \vec{c})$, v_o and v_d, the Multi-Objective Shortest Path Problem (MO-SPP) requires computing a maximal cost-unique Pareto-optimal set.

[1] Note that K is the length of the vector and M is the number of objective; K may be equal to M but does not necessarily have to.

[2] In the literature, other symbols such as $\preceq$, $\succeq$ and $\succ$ are also frequently used to denote the dominance relation between two vectors. We choose to use $\prec$ in this article since the goal here is to minimize costs.

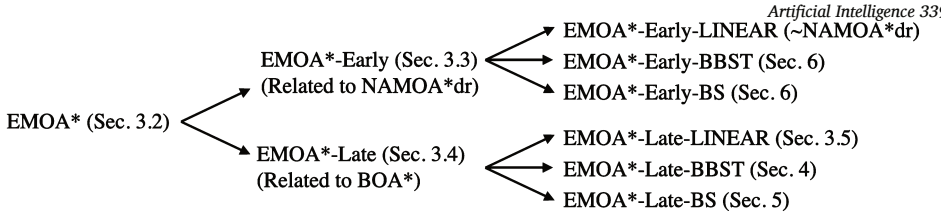

Fig. 2. The relationship of different approaches under the EMOA* framework.

Finally, we introduce a few notations to simplify the subsequent presentation. The cost vectors corresponding to a maximal cost-unique Pareto-optimal set is called the *Pareto-optimal front*. Given two vectors $\vec{a}$ and $\vec{b}$ of the same length K (with $K \geq 2$), we say that $\vec{a}$ *weakly dominates* $\vec{b}$ ($\vec{a} \preceq \vec{b}$), if every component in $\vec{a}$ is less than or equal to $\vec{b}$ (i.e., $\forall k \in \{1, 2, \dots, K\}$, $a_k \leq b_k$). Note that, $\vec{a} \preceq \vec{b}$ is equivalent to $\vec{a} \prec \vec{b}$ or $\vec{a} = \vec{b}$. Given two vectors $\vec{a}$ and $\vec{b}$ of the same length K (with $K \geq 2$), let $\vec{a} <_{\text{lex}} \vec{b}$ represent that $\vec{a}$ is lexicographically smaller than $\vec{b}$. Similarly, $>_{\text{lex}}, \leq_{\text{lex}}, \geq_{\text{lex}}$ can be defined between $\vec{a}$ and $\vec{b}$. Given a set B of vectors of the same length, B is called a set of cost-unique non-dominated vectors if any two vectors $a, b \in B$ satisfy that $a \not\prec b$, $b \not\prec a$ and $a \neq b$. Additionally, given a set B of vectors of the same length, let $\mathcal{ND}(B)$ denote the maximal cost-unique non-dominated subset of vectors in B. Finally, let $\text{Trunc} : \mathbb{R}^K \rightarrow \mathbb{R}^{K-1}$ denote a *truncation function* that removes the first component from the input vector, and we use $\text{Trunc}(\vec{a})$ to denote the truncated vector corresponding to $\vec{a}$.

3. Enhanced Multi-Objective A* (EMOA*) framework

This section begins by introducing the basic concepts and notations in Sec. 3.1 and then presents our EMOA* framework in Sec. 3.2. We then present two instantiations of the EMOA* framework in Sec. 3.3 and Sec. 3.4, depending on when to conduct the dominance checking during the search. For both instantiations, there are three key abstract procedures related to dominance checking, and we define the corresponding sub-problems that need to be solved by these procedures in Sec. 3.5. Finally, we discuss the properties of EMOA* in Sec. 3.6. The relationship of different approaches are summarized in Fig. 2.

3.1. Basic concepts

Let $l = (v, \vec{g})$ denote a *label*,[3] which is a tuple of a vertex $v \in V$ and a cost vector $\vec{g}$. A label represents a path from v_o to v with cost vector $\vec{g}$. To simplify the presentation, given a label l, let $v(l)$ and $\vec{g}(l)$ denote the vertex and the cost vector contained in label l, respectively. Two labels l, l' are comparable only when $v(l) = v(l')$, and a label l is said to be dominated by (or is equal to) another label l' if $\vec{g}(l) \prec \vec{g}(l')$ (or $\vec{g}(l) = \vec{g}(l')$).

Let $\vec{h}(v), v \in V$ denote a heuristic that estimates the cost-to-go from v to v_d. As mentioned, we limit ourselves to single-valued heuristics in this paper. If every component of $\vec{h}(v), v \in V$ is no larger than the corresponding component of the cost vector of any possible path from v to v_d, then $\vec{h}$ is referred to as an *admissible* heuristic. If a heuristic satisfies $\vec{h}(v) \leq \vec{h}(u) + \vec{c}(u, v), \forall u, v \in V$ (intuitively speaking, the triangle inequality), then $\vec{h}$ is called a *consistent* heuristic. A consistent heuristic is always admissible if $\vec{h}(v_d) = \vec{0}$, i.e., the heuristic vector of the goal vertex is a zero vector.

Throughout this paper we will use the so-called "ideal-point heuristic" $\vec{h}^{\text{ideal}}$ which combines a set of M single-objective heuristics $h_1, \dots, h_M$. Let $\pi_i^{ideal}(v, v_d)$ denote a minimum cost path from each vertex $v \in V$ to v_d according to the i-th objective, and let h_i denote the cost value of the path $\pi_i^{ideal}(v, v_d)$. Then, this ideal-point heuristic is defined as $\vec{h}^{\text{ideal}}(v) := (h_1(v), \dots, h_M(v))$. The ideal-point heuristic, which is consistent, is easily computed by running M single-objective instances of Dijkstra's algorithm starting backwards from the destination v_d to all other vertices in G, i.e., one instance for each objective. Within the EMOA* framework, specific instantiations require consistent heuristics in order to use some fast dominance checking techniques. For the rest of the paper, we assume the heuristic is consistent.

The $\vec{f}$-vector of a label l is defined as $\vec{f}(l) := \vec{g}(l) + \vec{h}(v(l))$ and let OPEN denote a priority queue of labels. At any time during the search, OPEN contains labels that will be either expanded or discarded in future iterations of the search. While there are many ways to order OPEN can be ordered (see [38]), here we limit the discussion to the setting where labels in OPEN are prioritized by their corresponding $\vec{f}$-vectors in lexicographic order from the minimum to the maximum. This is a widely used ordering scheme in MOA* search [11,19,24].

Additionally, the search needs to store the non-dominated paths from v_o to any other vertex $u \in V$. Let $parent(l)$ denote the parent pointer of label l that represents the label from which l is generated. By iteratively backtracking the parent pointers of l, a path from v_o to $v(l)$ can be reconstructed. Let $\mathcal{F}_{\text{open}}(u), u \in V$ denote a set of non-dominated labels l at vertex u (i.e., $v(l) = u$) that are in OPEN. In other words, labels in $\mathcal{F}_{\text{open}}(u)$ are generated by the search and are to be expanded or discarded in the future search. Correspondingly, let $\mathcal{F}_{\text{closed}}(u), u \in V$ denote a set of non-dominated labels at vertex u, which have been expanded during the search. Each label in $\mathcal{F}_{\text{closed}}(u), u \in V$ represents a non-dominated path from v_o to u. Furthermore, let $\mathcal{F}(u), u \in V$ denote the *front* set at vertex

[3] To identify a path, different names, such as nodes [11] and labels [36,37], have been used in the multi-objective path-planning literature. This article uses "labels" to identify paths and reserves "nodes" for the tree nodes in the balanced binary search tree in the ensuing section.

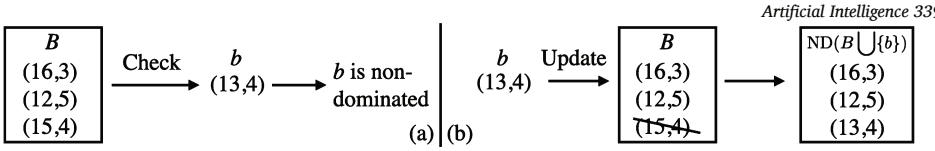

Fig. 3. Examples of the DC and NSU problems. (a) Visualization of a DC problem that requires checking if there exists a vector in a set B that weakly dominates the given vector b. In this example, b is non-dominated by any vectors in B. (b) Visualization of a NSU problem that requires computing $\mathcal{ND}(B \bigcup \{b\})$. In this example, $(15,4)$ is filtered from B since it is weakly dominated by b.

Algorithm 1 EMOA* Search Framework.

```
1: l_o ← (v_o, 0⃗), parent(l_o) ← NULL
2: Add l_o to OPEN
3: F(v) ← ∅, ∀v ∈ V
4: while OPEN ≠ ∅ do
5:     l ← OPEN.pop()                                   ▷ Label extracted
6:     if CheckUpdateBeforeExp (l) then
7:         continue
8:     if v(l) = v_d then
9:         continue
10:    for all v' ∈ GetSuccessors(v(l)) do              ▷ Label expanded
11:        l' ← (v', g⃗(l) + c⃗(v, v')), parent(l') ← l
12:        f⃗(l') ← g⃗(l') + h⃗(v(l'))
13:        if CheckUpdateAfterGen (l) then
14:            continue
15:        Add l' to OPEN
16: return F(v_d)
```

u, which stores non-dominated labels l at vertex u (i.e., $v(l) = u$) during the search. As we will see in Sec. 3.3 and 3.4, $\mathcal{F}(u)$ is the same as either $\mathcal{F}_{\text{open}}(u)$ or $\mathcal{F}_{\text{open}}(u) \cup \mathcal{F}_{\text{closed}}(u)$, depending on the specific instantiation of EMOA*. When presenting the framework EMOA* in Sec. 3, we use $\mathcal{F}(u)$ in order to unify different instantiations under a common framework. Finally, each label in $\mathcal{F}(v_d)$ identifies a solution.

As presented in the ensuing sections, EMOA* requires three procedures as building blocks: `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront`. These three procedures encapsulate the computation related to dominance checking, and the computational efficiency of these procedures affects the overall runtime of the search [11,19]. Here, we formally define the problems that need to be solved by these three procedures before presenting the algorithms.

Definition 3 *(Dominance Checking (DC) Problem).* Given a set B of K-dimensional cost-unique non-dominated vectors (with $K \geq 2$) and a new K-dimensional vector $\vec{b}$, the DC problem aims to verify whether there exists a vector $\vec{b}' \in B$ that weakly dominates $\vec{b}$, i.e., $\vec{b}' \preceq \vec{b}$.

Definition 4 *(Non-Dominated Set Update (NSU) Problem).* Given a set B of K-dimensional non-dominated vectors (with $K \geq 2$) and a new K-dimensional vector $\vec{b}$ that is non-dominated by any vector in B, the NSU problem computes $\mathcal{ND}(B \bigcup \{\vec{b}\})$.

Examples of the DC and NSU problems are provided in Fig. 3. The relationship between the aforementioned three procedures and these two problems can be described as follows.

- In `IsDomByFront`, given a label l and $\mathcal{F}(v(l))$, an equivalent DC problem can be generated with input $\vec{b} = \vec{g}(l)$ and $B = \{\vec{g}(l') | l' \in \mathcal{F}(v(l))\}$.
- In `IsDomBySol`, given a label l and $\mathcal{F}(v(l))$, an equivalent DC problem can be generated with $\vec{b} = \vec{f}(l)$ and $B = \{\vec{f}(l') | l' \in \mathcal{F}(v_d)\}$. Note that $\vec{f}(l') = \vec{g}(l'), \forall l' \in \mathcal{F}(v_d)$.
- In `FilterAndAddFront`, given a label l and $\mathcal{F}(v(l))$, an equivalent NSU problem can be generated with $\vec{b} = \vec{g}(l)$ and $B = \{\vec{g}(l') | l' \in \mathcal{F}(v(l))\}$.

3.2. The EMOA* search framework

As shown in Algorithm 1, to initialize the search (Lines 1-3), EMOA* first creates an initial label $l_o = (v_o, \vec{0})$, and sets its parent pointer to $NULL$, which means l_o has no parent. EMOA* then adds l_o to OPEN for future search (Table 1). Additionally, we assume that the entire graph is known and the front set at each vertex is initialized to be an empty set. In Algorithm 1, we say a label is *extracted* from OPEN, when EMOA* reaches Line 5. We say a label is *expanded*, when EMOA* reaches Line 10. During the search, the set of expanded labels is always a subset of the extracted labels. Additionally, we say a new label is *generated* when EMOA* reaches Line 11.

Table 1
Frequently used notations, procedure names and abbreviations.

Notation	Meaning
$G=(V,E)$	A graph G with vertex set V and edge set E.
l_o, l, l', l''	Labels.
$\vec{g}(l), \vec{h}(v(l)), \vec{f}(l)$	The g,h,f-vector related to label l.
$\mathcal{F}(v), \mathcal{F}_{\text{open}}(v), \mathcal{F}_{\text{closed}}(v)$	The frontier sets at vertex $v \in V$.
`CheckUpdateBeforeExp`	The check and update procedure before the expansion of a label.
`CheckUpdateAfterGen`	The check and update procedure after the generation of a label.
`IsDomBySol`	Check if a label is dominated by any solution found.
`IsDomByFront`	Check if a label l is dominated by any label in $\mathcal{F}(v(l))$.
`FilterAndAddFront`	Use a label l to filter $\mathcal{F}(v(l))$ and then add l to $\mathcal{F}(v(l))$.
DC	The Dominance Check Problem (Definition 3).
NSU	The Non-Dominated Set Update Problem (Definition 4).
BBST	Balanced Binary Search Tree.
BS	Binary Search.
EMOA*	A general multi-objective search framework.
EMOA*-Early	An instantiation of EMOA* framework with early dominance checking.
EMOA*-Late	An instantiation of EMOA* framework with late dominance checking.
EMOA*-Late-LINEAR	An algorithm that implements EMOA*-Late using simple approaches to solve the DC and NSU problems.
EMOA*-Late-BBST	An algorithm that implements EMOA*-Late using BBST-based approaches to solve the DC and NSU problems.
TOA*-Late-BBST	An improved version of EMOA*-Late-BBST when $M=3$.
EMOA*-Late-BS	An algorithm that implements EMOA*-Late using BS-based approaches to solve the DC and NSU problems.
TOA*-Late-BS	An improved version of EMOA*-Late-BS when $M=3$.

Algorithm 2 EMOA* with Early Dominance Checking (NAMOA*dr).

```
   CheckUpdateAfterGen(l')
 1: if IsDomByFront (l') or IsDomBySol (l') then
 2:     return true                                   ▷ l' is pruned.
 3: L ← FilterAndAddFront (l')
 4: RemoveOpen (L)
 5: return false                                      ▷ l' is not pruned.

   CheckUpdateBeforeExp(l)
 6: if IsDomBySol (l) then
 7:     remove l from F(v(l))
 8:     return true                                   ▷ l is pruned.
 9: return false                                      ▷ l is not pruned.
```

After the initialization, in each *expansion cycle* (Lines 4-15), the label with the lexicographic minimum $\vec{f}$-value is popped from OPEN and is denoted as l in Algorithm 1. EMOA* then conducts procedure `CheckUpdateBeforeExp` on Line 6, where label l is checked for dominance and is used to update $\mathcal{F}(v)$ if l is non-dominated. Specifically, if l is dominated in `CheckUpdateBeforeExp`, l is discarded and the current expansion cycle ends, because l cannot lead to a cost-unique Pareto-optimal solution. Otherwise (i.e., l is non-dominated), l is used to update $\mathcal{F}(v(l))$ in `CheckUpdateBeforeExp`. We will discuss two different implementations of `CheckUpdateBeforeExp` later, which lead to different search algorithms. Afterwards, label l is verified whether $v(l)=v_d$ (Line 8). If $v(l)=v_d$, the current expansion cycle ends; Otherwise, l is expanded, as explained next.

To expand a label l (i.e., to expand the path represented by label l), for each successor vertex v' of $v(l)$ in G, EMOA* creates a new label $l'=(v', \vec{g}(l)+\vec{c}(v,v'))$, which represents a new path from v_o to v' via $v(l)$ by extending l (i.e., extending the path represented by l). The parent pointer $parent(l')$ is set to l, which helps reconstruct the path represented by l' after the search terminates. Then, EMOA* conducts `CheckUpdateAfterGen` on Line 13, where the newly generated label l' is checked for dominance. If l' is non-dominated, depending on the implementation, `CheckUpdateAfterGen` will either use l' to update $\mathcal{F}(v(l'))$, or simply do nothing as discussed later. Finally, if l' is non-dominated in `CheckUpdateAfterGen`, l' is added to OPEN for future expansion.

In EMOA*, the search terminates when OPEN is empty. At termination, EMOA* returns $\mathcal{F}(v_d)$ (Line 17), which is a set of labels where each label represents a solution. This set of solutions is a maximal cost-unique Pareto-optimal set. The cost vectors of these solutions are the Pareto-optimal front.

3.3. EMOA* with early dominance check

The first instantiation of EMOA* is shown in Algorithm 2 and is referred to as EMOA*-Early. The search process of EMOA*-Early is similar to the search process of NAMOA* [33] and NAMOA*dr [19]. We discuss the relationship between them at the end of this section.

In EMOA*-Early, when a new label l' is generated on Line 11 in Algorithm 1, `CheckUpdateAfterGen` (Line 1-5 in Algorithm 2) first invokes the procedure `IsDomByFront` to compare l' against the existing labels $l'' \in \mathcal{F}(v(l'))$ and check if $\vec{g}(l')$ is weakly dominated by $\vec{g}(l'')$. Similarly, `CheckUpdateAfterGen` also calls the procedure `IsDomBySol` to compare l' against the existing solutions represented by label $l'' \in \mathcal{F}(v_d)$ to check if $\vec{f}(l')$ is weakly dominated by $\vec{f}(l'')$. If l' is dominated in either `IsDomByFront` or

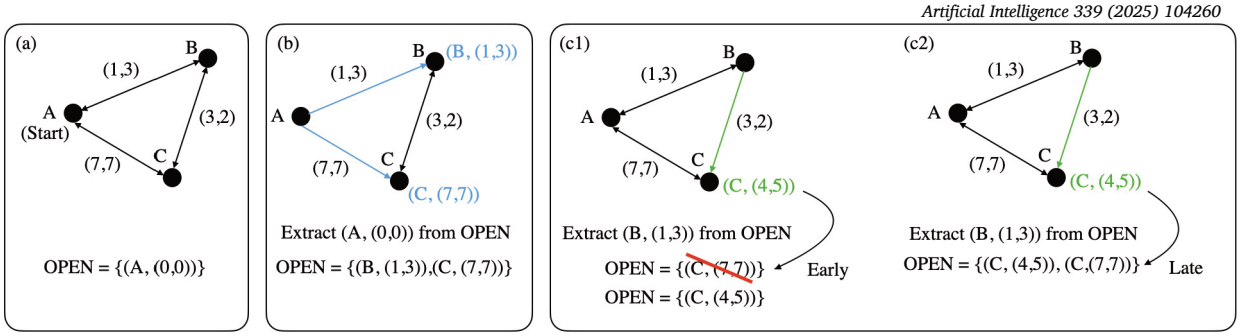

Fig. 4. Examples of early and late dominance checking. (a) shows the example and the initial label in OPEN after initialization. In this example, we assume the heuristics of all vertices are zero vectors (i.e., $\vec{f}(l) = \vec{g}(l)$ for any label l generated during the search). (b) shows that in the first iteration, label $(v = A, \vec{f} = (0,0))$ is extracted from OPEN and expanded, which generates two new labels that are added to OPEN. (c1) and (c2) show that the label $(v = B, \vec{f} = (1,3))$ is extracted from OPEN for expansion, and demonstrate the difference between Early Dominance Checking and Late Dominance Checking. In (c1), since the newly generated label $(v = C, \vec{f} = (4,5))$ dominates the existing label $(v = C, \vec{f} = (7,7))$, Early Dominance Checking removes $(v = C, \vec{f} = (7,7))$ from OPEN. In (c2), Late Dominance Checking does not remove $(v = C, \vec{f} = (7,7))$ from OPEN after the generation of $(v = C, \vec{f} = (7,7))$, and $(v = C, \vec{f} = (7,7))$ will be discarded in a future expansion cycle when $(v = C, \vec{f} = (7,7))$ is extracted from OPEN before expansion.

`IsDomBySol`, l' should be discarded and `CheckUpdateAfterGen` returns true. Otherwise (i.e., l' is non-dominated), `CheckUpdateAfterGen` first calls the procedure `FilterAndAddFront` (Line 3 in Algorithm 2) to use l' to *filter* $\mathcal{F}(v)$, where any existing label $l'' \in \mathcal{F}(v(l'))$ are removed if $\vec{g}(l')$ weakly dominates $\vec{g}(l'')$, then adds l' to $\mathcal{F}(v(l'))$, and finally returns false.

In EMOA*-Early, a non-dominated label l' is added to $\mathcal{F}(v(l'))$ after its generation. As a result, $\mathcal{F}(u), u \in V$ contains both labels that are in OPEN and labels that have been expanded (i.e., closed). In other words, $\mathcal{F}(u) = \mathcal{F}_{\text{open}}(u) \cup \mathcal{F}_{\text{closed}}(u)$. Therefore, when an existing label l is removed from $\mathcal{F}(u)$ in `FilterAndAddFront` on Line 3 in Algorithm 2, if l is in OPEN, the search should also remove l from OPEN to avoid expanding l in the future. This is achieved by `RemoveOpen` (L) on Line 4 in Algorithm 2, where L denotes the set of labels that are filtered in `FilterAndAddFront` on Line 3.

The existence of `RemoveOpen` ensures that at any time of the search, for any label $l \in$ OPEN, there does not exist another label $l' \in$ OPEN such that $\vec{g}(l')$ weakly dominates $\vec{g}(l)$ (Fig. 4). Therefore, in each expansion cycle, after a label l is extracted from OPEN, `CheckUpdateBeforeExp` (Lines 6-9 in Algorithm 2) does not need to compare l against the existing labels in $\mathcal{F}(v(l))$ any more. Instead, `CheckUpdateBeforeExp` only need to check if $\vec{f}(l)$ is weakly dominated by $\vec{f}(l_d)$ for some $l_d \in \mathcal{F}(v_d)$, where l_d represents an existing solution and $\vec{f}(l_d) = \vec{g}(l_d)$ since $\vec{h}(v_d) = 0$. If l, the extracted label from OPEN in the current expansion cycle, is dominated, then l is removed from $\mathcal{F}(v(l))$.

Remark 1. EMOA*-Early has the same search process as NAMOA* [33]. In the literature, NAMOA* has been expedited in NAMOA*dr by employing a technique called dimensionality reduction [19], which can be summarized as follows. With (i) a consistent heuristic and (ii) an OPEN priority queue where labels are prioritized in lexicographic order, then, the first component of the cost vectors corresponding to labels that are extracted from OPEN are guaranteed to be monotonically non-decreasing. The set of labels that are expanded at a vertex is a subset of all labels that are extracted from OPEN. Therefore, the first component of the cost vector of labels that are expanded at a vertex must be non-decreasing during the search and can be ignored for dominance checking. In other words, given a label l, to check if any existing label $l' \in \mathcal{F}_{\text{closed}}(v(l))$ weakly dominates l, NAMOA*dr only needs to check if the truncated vectors $\text{Trunc}(\vec{g}(l'))$ weakly dominates $\text{Trunc}(\vec{g}(l))$. This idea of truncating vectors for dominance checking is referred to as the "dimensionality reduction" in [19], which has been shown to speed up the search.

It is worthwhile to point out that, this dimensionality reduction is only applicable to the closed set, i.e., $\mathcal{F}_{\text{closed}}(u)$, at a vertex u, and is not applicable to the open set $\mathcal{F}_{\text{open}}(u)$. The reason is that: the first component of the cost vector of labels l that are added to $\mathcal{F}_{\text{open}}(v(l))$ during the search may not be monotonically non-decreasing, and the first component cannot be ignored for dominance checking. As a result, in EMOA*-Early, to employ the dimensionality reduction technique, the algorithm has to separate $\mathcal{F}(u)$ at a vertex $u \in V$ into $\mathcal{F}_{\text{closed}}(u)$ and $\mathcal{F}_{\text{open}}(u)$ so that the dimensionality reduction can be applied to $\mathcal{F}_{\text{closed}}(u)$.

In the next section, we present EMOA*-Late, the second implementation of EMOA*, which defers the dominance checking of a newly generated label l related to $\mathcal{F}_{\text{open}}(v(l))$ until l is about to be expanded, and is able to apply the dimensionality reduction technique to all dominance checking operations.

3.4. EMOA* with late dominance check

The second instantiation of EMOA* is shown in Algorithm 3 and we refer to it as EMOA*-Late. EMOA*-Late is similar to BOA* [11] and their relationship is discussed at the end of this section.

In EMOA*-Late, when a new label l' is generated on Line 11 in Algorithm 1, `CheckUpdateAfterGen` (i.e., Line 1-3 in Algorithm 3) first invokes both `IsDomByFront` and `IsDomBySol` to check if l' is dominated. If l' is dominated in either `IsDomByFront` or `IsDomBySol`, l' should be discarded and `CheckUpdateAfterGen` returns true. Otherwise, l' is non-dominated. In this case, EMOA*-Late directly adds l' to OPEN for future expansion, without invoking the procedure `FilterAndAddFront` in comparison to EMOA*-Early.

Algorithm 3 EMOA* with Late Dominance Checking (BOA*).

```
CheckUpdateAfterGen(l')
1: if IsDomByFront (l') or IsDomBySol (l') then
2:     return true                                   ▷ l' is pruned.
3: return false                                      ▷ l' is not pruned.

CheckUpdateBeforeExp(l)
4: if IsDomByFront (l) or IsDomBySol (l) then
5:     return true                                   ▷ l is pruned.
6: FilterAndAddFront (l)
7: return false                                      ▷ l is not pruned.
```

When a label l is extracted from OPEN, both `IsDomByFront` and `IsDomBySol` are called by `CheckUpdateAfterGen` on Line 4 to check if l is dominated. If l is dominated, l is simply discarded and the current expansion cycle ends. Otherwise, l is used to filter $\mathcal{F}(v(l))$ at first and then added to $\mathcal{F}(v(l))$.

In EMOA*-Late, $\mathcal{F}(u), u \in V$ only contains labels that are extracted from OPEN and never contains labels that are generated and in OPEN. In other words, $\mathcal{F}(u) = \mathcal{F}_{\text{closed}}(u)$ during the search in EMOA*-Late. As a result, EMOA*-Late avoids the operations on Line 4 in Algorithm 2 in EMOA*-Early; and EMOA*-Late defers the operations on Line 3 in Algorithm 2 in EMOA*-Early until a label is extracted from OPEN before being expanded (Fig. 4). In practice, $\mathcal{F}_{\text{open}}(u), u \in V$ is often a large set, and it can be computationally expensive to run dominance checking against $\mathcal{F}_{\text{open}}(u)$ every time a new label is generated. As shown in BOA* [11], this modification of the search also allows BOA* to implement the `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront` procedures as a constant-time operation in the presence of two objectives.

Remark 2. EMOA*-Late is similar to BOA* [11] with the only difference that EMOA*-Late introduces the `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront` procedures to encapsulate some common operations on $\mathcal{F}(v), v \in V$ that are related to dominance checking. Within the EMOA*-Late framework, BOA* can be regarded as an instantiation of the framework when there are two objectives.

We briefly summarize BOA* as follows. BOA* leverages the aforementioned dimensionality reduction in NAMOA*dr. Since there are two objectives only, after the dimensionality reduction, the cost vector of a label becomes a scalar value. Correspondingly, to represent $\mathcal{F}(v), v \in V$, BOA* only needs to store the minimum scalar value (denoted as $g_2^{\min}(v)$) among labels that are expanded at v during the search. As a result, `IsDomByFront`, `IsDomBySol` for a label l can be implemented by comparing the second component of $\vec{g}(l)$, i.e., $g_2(l)$, against $g_2^{\min}(v(l))$, which is a constant-time operation, and `FilterAndAddFront` can be implemented by updating $g_2^{\min}(v(l))$ with $g_2(l)$ when $g_2(l) < g_2^{\min}(v(l))$, which is also a constant-time operation.

It has been shown that when there are two objectives ($M = 2$), BOA* runs faster than NAMOA*dr in general due to the late dominance check and the resulting constant dominance checking operations [11].

3.5. Solving the check and update problems

Solving DC and NSU problems with different approaches within the EMOA* framework lead to different algorithms. As an example, a simple approach that solves the DC problem iterates each vector $\vec{b}' \in B$ and check if $\vec{b}' \preceq \vec{b}$, which has a runtime complexity of $O(|B| \cdot K)$. A simple method that solves the NSU problem takes two steps: (i) filter B by removing from B all vectors that are weakly dominated by $\vec{b}$, and (ii) add $\vec{b}$ into B. Here, a simple method for step (i) needs to iterate the vectors in B in order to remove all dominated vectors and has a runtime complexity of $O(|B| \cdot K)$, and step (ii) takes constant time. Consequently, the overall runtime complexity is $O(|B| \cdot K)$. With these simple approaches to solve the DC and NSU problems, for either EMOA*-Early or EMOA*-Late, a corresponding search algorithm is determined. For EMOA*-Late, we call the corresponding algorithm EMOA*-Late-LINEAR, where LINEAR means the algorithm uses a linear scan of vectors in B for dominance checking. For EMOA*-Early, the resulting algorithm is NAMOA* [33], which does not use the dimensionality reduction technique in comparison to NAMOA*dr.

Remark 3. We are now ready to revisit the aforementioned Kung's method [20] and discuss its relationship to this article. Kung's method aims to solve the following problem (hereafter referred to as Kung's problem for simplicity). Given an arbitrary set B of K-dimensional vectors ($K \geq 2$), Kung's problem seeks to compute $\mathcal{ND}(B)$. We refer the reader to [20] for more detail about Kung's method and the runtime complexity. The DC and NSU problems can be regarded as *incremental* versions of Kung's problem: specifically, after a new vector $\vec{b}$ is generated, $\vec{b}$ is checked for dominance against the existing vectors in B, which is a DC problem; then, if $\vec{b}$ is non-dominated, B is updated as $\mathcal{ND}(B \bigcup \{\vec{b}\})$, which is a NSU problem. This *incremental* formulation of Kung's problem is important to EMOA*, since $\mathcal{F}(v)$ at each vertex $v \in V$ is constructed in an incremental manner during the search.

3.6. Theoretical properties of EMOA*

EMOA*-Early has the same search process as NAMOA* [33] (and NAMOA*dr [19]). As a result, EMOA*-Early has the same properties as NAMOA* as long as the three procedures `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront` are correctly implemented to solve the corresponding DC and NSU problems. The analysis of EMOA*-Early is thus omitted. We hereafter analyze only the properties of EMOA*-Late.

A MO-SPP instance is *feasible* if there is at least one solution, and otherwise (i.e., there is no path from v_o to v_d in G), the instance is infeasible. With an implementation of the three procedures `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront` that correctly solve the corresponding DC and NSU problems, it is guaranteed that EMOA*-Late terminates in finite time for both feasible and infeasible instances (Theorem 1). EMOA*-Late computes a maximal cost-unique Pareto-optimal set at termination for feasible instances (Theorem 2).

We say a path $\pi = (v_1, v_2, \cdots, v_k)$ forms a loop at v_k, if v_k is the first vertex in π such that $v_k = v_p$ for some $p = 1, 2, \cdots, k-1$.

Lemma 1. *EMOA*-Late never adds a label l to OPEN if l represents a path that forms a loop at $v(l)$.*

Proof. We only need to consider the case where l is generated (Line 11) and before being added to OPEN (Line 15). (If l is not generated, l cannot be added to OPEN.) Let l denote a label representing a path π that forms a loop at $v(l)$, and let l' denote the label representing the corresponding path with the loop in π removed. Since l represents a path that forms a loop at $v(l)$, by definition, we know that $v(l) = v(l')$. The cost vector of edges in G are non-negative, and π has an additional loop in comparison with π', therefore, $\vec{g}(l)$ is weakly dominated by $\vec{g}(l')$. Furthermore, when l is generated, all of its parent labels from l to l_o must have been expanded. Therefore, l' must have been expanded and must have been added to $\mathcal{F}(v(l'))$. As a result, l is discarded at Line 13 since $v(l) = v(l')$ and $\vec{g}(l') \preceq \vec{g}(l)$. EMOA*-Late thus never adds l to OPEN. □

Theorem 1. *EMOA*-Late terminates in finite time for both feasible and infeasible instances.*

Proof. Since G is finite, there is a finite number $|\Pi|$ of paths without loops from v_o to any other vertex $v \in V$ in G. Based on Lemma 1, we know that, the maximum possible number of labels in OPEN is no larger than $|\Pi|$ during the search. EMOA*-Late eventually extracts all labels from OPEN and therefore terminates in finite time. The proof holds for both feasible and infeasible instances. □

Theorem 2. *EMOA*-Late computes a maximal cost-unique Pareto-optimal set at termination for feasible instances.*

Proof. In each expansion cycle (Lines 4-15 in Algorithm 1), EMOA*-Late extracts a label l from OPEN, whose $\vec{f}$-vector is the lexicographic minimum in OPEN. It means none of the remaining labels in OPEN can dominate l. With procedures `IsDomByFront` and `IsDomBySol`: label l is discarded if and only if it is weakly dominated by some other expanded labels. This implies that it can not lead to a cost-unique Pareto-optimal solution. If label l is not discarded, it is then added to $\mathcal{F}(v(l))$ after filtering $\mathcal{F}(v(l))$ using l, which ensures that $\mathcal{F}(v(l))$ contains only cost-unique non-dominated labels after the filtering. When Algorithm 1 terminates, each of the labels in $\mathcal{F}(v_d)$ must represent a cost-unique Pareto-optimal solution. Finally, when a label l is expanded, all possible successor labels of l are generated and the non-dominated ones are inserted into OPEN for future expansion. The algorithm terminates only when all labels are either expanded or discarded, which guarantees that a maximal set of cost-unique Pareto-optimal solutions are found. □

From now on, we focus on how to correctly implement the three procedures while achieving high computational efficiency. We begin with a BBST-based approach in Sec. 4 and a BS-based approach in Sec. 5. Both approaches leverage the dimensionality reduction technique and thus require consistent heuristics.

4. The EMOA*-Late-BBST Algorithm

This section presents an algorithm that implements the EMOA*-Late by leveraging balanced binary search trees (BBSTs). We review BBSTs in Sec. 4.1 and then elaborate the BBST-based approaches that solve the DC and NSU problems in Sec. 4.2 and Sec. 4.3 respectively. We finally describe the EMOA*-Late-BBST algorithm in Sec. 4.4.

4.1. Balanced Binary Search Trees (BBSTs)

Let n denote a node within a binary search tree (BST) with the following attributes:

- $n.key$ is the key of n, which is a K-dimensional vector. To compare two nodes, their keys are compared using the lexicographic order.
- $n.height$ is the height of n, which is the number of edges along the longest downwards path between n and a leaf node. A leaf node has a height of zero. The height of the root node is also called the height of the BST.
- $n.left$ and $n.right$ are the left child and the right child of n representing the left sub-tree and the right sub-tree, respectively.
- We say that $n = NULL$ if n does not exist in the BST. For example, if n is a leaf node, then $n.left = NULL$ and $n.right = NULL$.

We limit our focus to the AVL-tree, a popular balanced BST data structure. For any node n in an AVL-tree, let $d(n) := n.left.height - n.right.height$ denote the difference between the height of the left and right children. The AVL-tree is called balanced if $d(n) \in \{-1, 0, 1\}$. To maintain balance at insertion or deletion of nodes, an AVL-tree invokes the so-called rotation operations when $|d(n)| \geq 2$

Algorithm 4 `BBST-Check` $(n, \vec{b})$.

INPUT: n is a node in an AVL-tree and $\vec{b}$ is a vector
1: **if** $n = NULL$ **then**
2: **return** false
3: **if** $n.key \preceq \vec{b}$ **then**
4: **return** true
5: **if** $\vec{b} <_{\text{lex}} n.key$ **then**
6: **return** `BBST-Check` $(n.left, \vec{b})$
7: **else** ▷ i.e., $\vec{b} >_{\text{lex}} n.key$
8: **if** `BBST-Check` $(n.left, \vec{b})$ **then** ▷ Removed in TOA*-Late-BBST
9: **return** true
10: **return** `BBST-Check` $(n.right, \vec{b})$

Algorithm 5 `BBST-Filter` $(n, \vec{b})$.

INPUT: n is a node in an AVL-tree and $\vec{b}$ is a vector
1: **if** $n = NULL$ **then**
2: **return** $NULL$
3: **if** $\vec{b} >_{\text{lex}} n.key$ **then**
4: $n.right \leftarrow$`BBST-Filter` $(n.right, \vec{b})$
5: **else**
6: $n.left \leftarrow$`BBST-Filter` $(n.left, \vec{b})$
7: $n.right \leftarrow$`BBST-Filter` $(n.right, \vec{b})$
8: **if** $\vec{b} \preceq n.key$ **then**
9: **return** *AVL-Delete*(n)
Note: the tree needs to be re-balanced after the entire filtering process.

in order to re-balance the tree. Consequently, given an AVL-tree of size N (i.e., containing N nodes), the height of the root node is bounded by $O(\log N)$.

4.2. BBST-based checking method

Given a set B of cost-unique non-dominated vectors, let $\mathcal{T}_B$ denote an AVL-tree that stores all vectors in B as the keys of tree nodes. Given a new vector $\vec{b}$, the DC problem can be solved via Algorithm 4, which traverses the tree recursively while running dominance checking.

Algorithm 4 is invoked with `BBST-Check` $(\mathcal{T}_B.root, \vec{b})$, where $\mathcal{T}_B.root$ is the root node of the tree and $\vec{b}$ is an input vector to be checked for dominance. As the base case (Line 1), if the input node n is $NULL$, the algorithm terminates and returns false, which means $\vec{b}$ is non-dominated. When the input node is not $NULL$, $\vec{b}$ is checked for dominance against $n.key$ and returns true if $n.key \preceq \vec{b}$. Otherwise, the algorithm verifies if $\vec{b}$ is lexicographically smaller than $n.key$.

Case 1 If $\vec{b} <_{\text{lex}} n.key$, there is no need to traverse the right sub-tree of n, since any node in the right sub-tree of n must be lexicographically larger than n and thus cannot weakly dominate b. The algorithm then recursively invokes itself to traverse only the left sub-tree for dominance checking.

Case 2 Otherwise (i.e., $\vec{b} >_{\text{lex}} n.key$), the algorithm first invokes itself to traverse the left sub-tree (Line 8) and then the right sub-tree (Line 10) for dominance checking. Note that, in this case, both child nodes need recursive traversal to ensure correctness.

4.3. BBST-based update method

Similarly, given a set of non-dominated vectors B that is stored as a BBST $\mathcal{T}_B$ and a non-dominated vector $\vec{b}$, the NSU problem can be solved by (i) invoking Algorithm 5 to remove nodes from the tree $\mathcal{T}_B$ whose keys are dominated by $\vec{b}$ and (ii) inserting the input (non-dominated) vector $\vec{b}$ into the tree. Here, step (ii) is a regular AVL-tree insertion operation, which takes $O(\log |B|)$ time, and we will focus on step (i) in the ensuing paragraphs.

For step (i), Algorithm 5 is invoked with `BBST-Filter` $(\mathcal{T}_B.root, \vec{b})$, where $\mathcal{T}_B.root$ is the root node of the tree. As shown in Algorithm 5, as the base case (Line 1), if the input node is $NULL$, the algorithm terminates and returns $NULL$. When the input node n is not $NULL$, the algorithm verifies whether $\vec{b} >_{\text{lex}} n.key$.

Case 1 If $\vec{b} >_{\text{lex}} n.key$, there is no need to filter the left sub-tree of n, since any node in the left sub-tree of n must be non-dominated by $\vec{b}$. The algorithm recursively invokes itself to only traverse the right sub-tree for filtering.

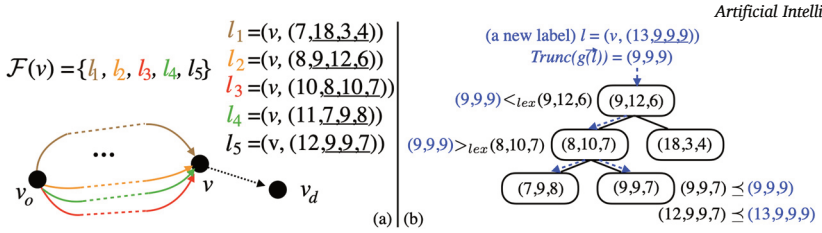

Fig. 5. (a) A visualization of $\mathcal{F}(v)$ at some vertex v in the graph G during the EMOA* search. Here, there are five labels in $\mathcal{F}(v)$. The underlined three numbers of each $\vec{g}$-vector indicate the corresponding truncated vector $\text{Trunc}(\vec{g})$ (as defined in Sec. 4.4). (b) The corresponding balanced binary search tree. The keys of the nodes in this tree form the non-dominated subset of the truncated vectors. The dashed blue arrows show the sequence of tree nodes that are traversed when running the `IsDomByFront` procedure (Algorithm 4). The truncated vector $(9,9,7)$ in the tree weakly dominates the input vector $(9,9,9)$, which indicates that the new label l (in blue) with $\vec{g}$-vector $(13,9,9,9)$ is weakly dominated and should be discarded.

Case 2 Otherwise (i.e., $\vec{b} <_{\text{lex}} n.key$),[4] the algorithm first invokes itself to traverse the left sub-tree (Line 6) and then the right sub-tree (Line 7) for filtering. Note that, in this case, both child nodes need to be traversed for further dominance checking to ensure correctness.

At the end (Line 8), $n.key$ is checked for dominance against $\vec{b}$. If $n.key$ is dominated, n is removed from the tree. After invoking Algorithm 5, if nodes are deleted, the tree needs to be re-balanced. Specifically, the tree can become unbalanced after running Algorithm 5 where the height difference between the left sub-tree and right sub-tree of a node n can be greater than 2, i.e., $|d(n)| \geq 2$. In this case, a single rotation operation related to n cannot re-balance the tree, and one possible implementation to re-balance the tree is to first mark the node to be deleted during the filtering process as described in Algorithm 5, and then conduct an in-order traversal of the tree, while skipping the marked nodes, to rebuild an AVL-tree. This implementation takes $O(|B| \cdot K)$ time with respect to the size of the tree. For the filtering part (Algorithm 5), in the worst case, the entire tree is traversed and all nodes in the tree are to be deleted (from the leave nodes to the root node), which takes $O(|B| \cdot K)$ time.

Theoretically, both Algorithm 4 and 5 run in $O(|B| \cdot K)$ time in the worst case, which is the same as the aforementioned linear approaches (i.e., iterates the vectors in B) in Sec. 3.5. However, as shown in our experimental results in Sec. 6, the BBST-based methods can solve the DC and NSU problems more efficiently in practice. The intuitive reason behind such efficiency is that, the AVL-tree is organized based on the lexicographic order, which can provide guidance when traversing the tree for dominance checking. As a result, only a small portion of the tree is traversed. Finally, note that the method in this section does not put any restriction on K.

4.4. EMOA* with BBST-based checking and update

This section elaborates how the aforementioned BBST-based algorithms (Algorithm 4 and 5) are used within the EMOA* framework (Algorithm 1). We refer to the resulting algorithm as EMOA*-Late-BBST. EMOA*-Late-BBST leverages the idea of dimensionality reduction as in NAMOA*-dr, which can expedite the BBST-based checking and update by reducing the length of the cost vectors by one.

Specifically, when the heuristic is consistent, and all labels are extracted from OPEN based on the lexicographic order of their $\vec{f}$-vectors from the minimum to the maximum, the sequence of labels being extracted at the same vertex has non-decreasing f_1 values, where f_1 represents the first component of the $\vec{f}$-vector of a label. Additionally, since all labels at the same vertex v have the same $\vec{h}$-vector, the sequence of labels being extracted at the same vertex also has non-decreasing g_1 values, where g_1 represents the first component of the $\vec{g}$-vector of a label. During the search of EMOA*-Late-BBST, when a new label l is generated, `IsDomByFront` only needs to perform dominance checking between $\text{Trunc}(\vec{g}(l))$ and $\text{Trunc}(\vec{g}(l')), \forall l' \in \mathcal{F}(v(l))$, instead of comparing $\vec{g}(l)$ with $\vec{g}(l'), \forall l' \in \mathcal{F}(v(l))$. Consequently, in EMOA*-Late-BBST, for each vertex $v \in V$, a BBST tree $\mathcal{T}_B$ is constructed with $B = \mathcal{ND}(\{\text{Trunc}(\vec{g}(l')), \forall l' \in \mathcal{F}(v)\})$ as aforementioned. In other words, the key of nodes in $\mathcal{T}_B$ forms a maximal cost-unique non-dominated subset of the truncated cost vector of labels in $\mathcal{F}(v)$.

To run `IsDomByFront` for a label l that is extracted from OPEN (Line 6 in Algorithm 1), `BBST-Check` $(n, \vec{b})$ in Algorithm 4 is invoked with $\vec{b} = \text{Trunc}(\vec{g}(l))$ and n being the root node of the tree $\mathcal{T}_B$. We provide a toy example for `IsDomByFront` in Fig. 5. Similarly, for `IsDomBySol` (Line 6 in Algorithm 1), `BBST-Check` $(n, \vec{b})$ in Algorithm 4 is invoked with $\vec{b} = \text{Trunc}(\vec{f}(l))$ and n being the root node of the tree $\mathcal{T}_{B'}$ with $B' = \mathcal{ND}(\{\text{Trunc}(\vec{g}(l')), \forall l' \in \mathcal{F}(v_d)\})$ (i.e., the set of all non-dominated truncated vectors of labels in the front set at the destination node). During the search, when a label l is extracted from OPEN and is used to update the front set in the procedure `FilterAndAddFront` (Line 8 in Algorithm 1), `BBST-Filter` $(n, \vec{b})$ in Algorithm 5 is first invoked with $\vec{b} = \text{Trunc}(\vec{g}(l))$ and n being the root node of the tree $\mathcal{T}_B$ where $B = \mathcal{ND}(\{\text{Trunc}(\vec{g}(l')), \forall l' \in \mathcal{F}(v(l))\})$. Then, $\vec{b} = \text{Trunc}(\vec{g}(l))$ is added to $\mathcal{T}_B$.

In summary, in EMOA*-Late-BBST, procedures `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront` only need to operate on the truncated vectors of labels, instead of the original vectors.

[4] Note that it's impossible to have $\vec{b} = n.key$: Within EMOA* (Algorithm 1), `FilterAndAddFront` is always invoked after `IsDomByFront`. If we have that $\vec{b} = n.key$, the `IsDomByFront` removes it and `FilterAndAddFront` will not be invoked (Line 6-8 in Algorithm 1).

Table 2
Runtime complexity of related methods as a function of the dimension M, and the size B of the AVL tree. Note that BOA* is a special case of EMOA* when $M = 2$, and TOA*-Late-BBST is an improved version of EMOA*-Late-BBST when $M = 3$.

	BOA*	TOA*-Late-BBST	EMOA*-Late-BBST
M	$= 2$	$= 3$	≥ 2
Dominance Checking Problem	$O(1)$	$O(\log \lvert B \rvert)$	$O(\lvert B \rvert \cdot (M-1))$
Non-Dominated Set Update Problem	$O(1)$	$O(\lvert B \rvert)$	$O(\lvert B \rvert \cdot (M-1))$

4.5. Discussion—EMOA*-Late-BBST as a generalization of BOA*

EMOA*-Late-BBST generalizes BOA* in the following sense. When $M = 2$, for any cost vector $\vec{g}$ of a label, the truncated vector $\text{Trunc}(\vec{g})$ is of length one and is thus a scalar value. In this case, the AVL-tree corresponding to $\mathcal{F}(v)$ of any vertex $v \in V$ in EMOA*-Late-BBST becomes a singleton tree: a tree with a single root node $\mathcal{T}_B.root$. The key value of $\mathcal{T}_B.root$ is the minimum value of $g_2(l)$ among all labels $l \in \mathcal{F}(v)$, which is the same as the auxiliary variable $g_2^{\min}$ introduced at each vertex in BOA*. Solving a DC problem requires only a scalar comparison between $\mathcal{T}_B.root.key$ and the scalar $\text{Trunc}(\vec{g})$, i.e., the truncated cost vector of the label selected from OPEN in each search iteration. Clearly, this scalar comparison takes constant time. Additionally, the `FilterAndAddFront` in EMOA*-Late-BBST requires simply assigning the scalar $\text{Trunc}(\vec{g})$ to $\mathcal{T}_B.root.key$ (i.e., $g_2^{\min}$), which also takes constant time. Therefore, BOA* is a special case of EMOA* when $M = 2$.

4.6. BBST-Based Tri-Objective A* (TOA*-Late-BBST)

When $M = 3$, Algorithm 4 can be further improved to achieve better theoretic runtime complexity, which then further expedites EMOA*-Late-BBST. We name this improved algorithm TOA*-Late-BBST (Tri-Objective A*), which differs from the EMOA*-Late-BBST by removing Lines 8-9 in Algorithm 4. In other words, when $M = 3$, each truncated vector $\vec{b}$ as well as the key of all nodes in the tree $\mathcal{T}_B$ have length $(M-1) = 2$. In this case, in Algorithm 4, when $\vec{b} >_{\text{lex}} n.key$ (i.e., Line 8 in Algorithm 4), there is no need to further traverse the left sub-tree. This property can be formally stated via the following theorem.

Theorem 3. *Given a two-dimensional vector $\vec{b}$, and an arbitrary node n in $\mathcal{T}_B$ with B denoting a set of cost-unique non-dominated two-dimensional vectors, if (i) $n.key$ neither dominates nor is equal to $\vec{b}$ and (ii) $\vec{b} >_{\text{lex}} n.key$, then the key of any nodes in the left sub-tree of n cannot dominate $\vec{b}$.*

Proof. Recall that we use subscripts to denote the specific component of a vector. From (i) and (ii), we know that $b_1 > n.key_1$ and $b_2 < n.key_2$. For any node n' in the left sub-tree of n, by construction of the tree, $n' <_{\text{lex}} n$ and thus $n'.key_1 \leq n.key_1$. Additionally, by definition, the key of every pair of nodes in $\mathcal{T}_B$ are non-dominated and non-equal to each other, thus $n'.key_2 > n.key_2$. Combining these together, we know that

$$b_2 < n.key_2 < n'.key_2.$$

Thus, $\vec{b}$ is not dominated by $n'.key$. Since n' can be any node in the left sub-tree of n, the theorem is hence proved. □

In TOA*-Late-BBST, the modified version of Algorithm 4 traverses the AVL tree either to the left sub-tree (when $\vec{b} <_{\text{lex}} n.key$) or to the right sub-tree (when $\vec{b} >_{\text{lex}} n.key$), which leads to a runtime complexity of $O(\log |B|)$ (note that $K = M - 1 = 2$ is a constant number and is thus omitted from $O(\log |B| \cdot K)$). We say that TOA*-Late-BBST is an improved version of EMOA*-Late-BBST when $M = 3$ because the theoretic runtime complexity is reduced from $O(|B|)$ to $O(\log |B|)$. Finally, we summarize the runtime complexity of solving DC and NSU problems in both the existing BOA* [11] and our algorithms (TOA*-Late-BBST and EMOA*-Late-BBST) in Table 2.

5. The EMOA*-Late-BS Algorithm

This section provides a different implementation of the three procedures in EMOA*-Late by (i) using a lexicographically sorted list to represent $\mathcal{F}(v)$ at a vertex $v \in V$, instead of using a BBST as in the previous section, and (ii) using a binary search to conduct fast dominance checking. We begin with the general case, i.e., an arbitrary number of objectives, in Sec. 5.1, and name this new approach as EMOA*-Late-BS, where BS stands for Binary Search. We then present the approach when there are three objectives in Sec. 5.2, and we refer to this approach as TOA*-Late-BS. TOA*-Late-BS (and EMOA*-Late-BS) have the same worst-case runtime complexity as TOA*-Late-BBST (and EMOA*-Late-BBST) as shown in Table 2. However, this BS-based approach can still be regarded as an improved version of the BBST-based approach since rotation operations that are required to maintain BBSTs are saved.

Algorithm 6 `BS-Check` $(B, \vec{b})$.

INPUT: B is a lexicographically sorted list of vectors; $\vec{b}$ is a vector to be checked for dominance.
1: run `BinarySearch` to find $\vec{b}^{\max}$, the lexicographically largest vector in B that is no larger than $\vec{b}$; let B' denote the list of vectors from the beginning of B till $\vec{b}^{\max}$.
2: **if** $\vec{b}^{\max}$ does not exist **then**
3: **return** false ▷ $\vec{b}$ is non-dominated.
4: **for all** $\vec{b}' \in B'$ **do**
5: **if** $\vec{b}' \preceq \vec{b}$ **then**
6: **return** true ▷ $\vec{b}$ is dominated.
7: **return** false ▷ $\vec{b}$ is non-dominated.

Algorithm 7 `BS-Filter` $(B, \vec{b})$.

INPUT: B is a lexicographically sorted list of vectors and $\vec{b}$ is a vector used to filter B.
1: run `BinarySearch` to find $\vec{b}^{\max}$, the lexicographically largest vector in B that is no larger than $\vec{b}$; let B' denote the list of vectors starting after $\vec{b}^{\max}$ till the end of B.
2: **if** $\vec{b}^{\max}$ does not exist **then**
3: $B' \leftarrow B$.
4: **for all** $\vec{b}' \in B'$ **do**
5: **if** $\vec{b} \preceq \vec{b}'$ **then**
6: remove $\vec{b}'$ from B.
7: **return** B.

5.1. EMOA*-Late-BS

EMOA*-Late-BS also leverages the dimensionality reduction technique. Here, for every $v \in V$, $\mathcal{F}(v)$ stores the truncated vectors, which are of length $M-1$. EMOA*-Late-BS represents the $\mathcal{F}(v)$ at each vertex $v \in V$ as a lexicographically sorted list B from the minimum to the maximum,[5] and runs a binary search over B to realize `IsDomByFront` which checks if an input vector b is weakly dominated by any existing vector in B. We show this BS-based checking procedure in Algorithm 6. Specifically, Algorithm 6 first uses `BinarySearch` (Line 1) to find the lexicographically largest vector $\vec{b}^{\max}$ in B that is lexicographically no larger than $\vec{b}$, which has a runtime complexity of $O(\log(|B|))$ in the worst case. Note that any vector in B that is lexicographically larger than $\vec{b}^{\max}$ is also lexicographically larger than $\vec{b}$ and thus cannot dominate $\vec{b}$. Let B' denote the list of vectors from the beginning of B till $\vec{b}^{\max}$. Algorithm 6 iterates B' to check if any existing vector in B' weakly dominates the given $\vec{b}$. As an edge case, if no such a $\vec{b}^{\max}$ is found during `BinarySearch` (Line 2), all vectors in B are lexicographically larger than $\vec{b}$ and no vector in B can dominate $\vec{b}$. In the worst case, $B' = B$ and the iteration of B' has a runtime complexity of $O(|B|)$. As a result, the entire Algorithm 6 has a worst-case runtime complexity of $O(|B|)$. Finally, to realize `IsDomBySol`, EMOA*-Late-BS compares the given vector $\vec{b}$ against the vectors in B_{v_d}, where B_{v_d} is a set of cost-unique non-dominated truncated cost vectors corresponding to labels in $\mathcal{F}(v_d)$, with the exactly same algorithm shown in Algorithm 6.

We have presented how to implement `IsDomByFront` and `IsDomBySol`, and we now present the implementation of `FilterAndAddFront` (Line 8 in Algorithm 1) using binary search. To update B with a new non-dominated vector $\vec{b}$, EMOA*-Late-BS begins by filtering B and then adds $\vec{b}$ into B to compute $\mathcal{ND}(B \bigcup \{\vec{b}\})$. We elaborate the filtering procedure in Algorithm 7. To filter B with $\vec{b}$, Algorithm 7 first runs `BinarySearch` over B to find the lexicographically largest vector $\vec{b}^{\max}$ that is lexicographically no larger than $\vec{b}$ (Line 1). In B, any vector that is lexicographically smaller than $\vec{b}^{\max}$ cannot be dominated by $\vec{b}$. Let B' denote the list of vectors starting after $\vec{b}^{\max}$ till the end of B. Algorithm 7 then iterates B' (Line 4) and removes any existing vector in B' that is dominated by $\vec{b}$. As a special case, if $\vec{b}_{\max}$ does not exist, Algorithm 7 iterates the entire B (Line 3).

After running Algorithm 7 to filter B (which takes $O(|B|)$ time), the vector $\vec{b}$ is added to B at the position immediately after $\vec{b}^{\max}$ in B so that B is still lexicographically sorted after the insertion. For the special case where $\vec{b}^{\max}$ does not exist, i.e., no vector in B is lexicographically smaller than $\vec{b}$, $\vec{b}$ is added to the beginning of B, since b is the lexicographically minimum vector in B. Overall, this implementation of `FilterAndAddFront` has a worst-case runtime complexity of $O(|B|)$.

Intuitively, both the BS-based approach in this section and the BBST-based approach in the previous section hinge on running a binary search for fast dominance checking. However, the BBST-based approach relies on a binary tree, which requires rotation operations to re-balance the trees when vectors are added or deleted, while the BS-based approach directly operates on a lexicographically sorted list and can bypass the rotation operations needed by the BBST.

5.2. TOA*-Late-BS

When there are only three objectives ($M = 3$), the BS-based checking procedure in EMOA*-Late-BS can be further improved, which leads to the algorithm TOA*-Late-BS that is outlined in Algorithm 8. Specifically, Algorithm 8 first runs a binary search (Line 1) to find the lexicographically largest vector $\vec{b}^{\max}$ in B that is lexicographically no larger than $\vec{b}$, which takes $\log(|B|)$ time in the worst case. This vector $\vec{b}^{\max}$ has the property that $b_1^{\max} \leq b_1$, since $\vec{b}^{\max}$ is lexicographically no larger than $\vec{b}$. To check if $\vec{b}$ is dominated or not, we now only need to check if $b_2^{\max} \leq b_2$ (this will be explained in the next paragraph). If so (Line 4), $\vec{b}$ is weakly

[5] Strictly speaking, each vertex v is associated with its own lexicographically sorted list B, thus we should write $B(v)$. However, to simplify notation, we omit v.

Algorithm 8 `BS-Check-TOA*` $(B, \vec{b})$.

INPUT: B is a lexicographically sorted list of two-dimensional vectors and $\vec{b}$ is two-dimensional vector to be checked for dominance.
1: run `BinarySearch` to find $\vec{b}^{\max}$, the largest vector in B that is no larger than $\vec{b}$.
2: **if** $\vec{b}^{\max}$ does not exist **then**
3: **return** false
4: **if** $b_2^{\max} \leq b_2$ **then**
5: **return** true
6: **else**
7: **return** false

dominated by $\vec{b}^{\max}$ and should be discarded; otherwise (Line 6), $\vec{b}$ is non-dominated by any existing vector in B. As an edge case (Line 2), if no such a $\vec{b}^{\max}$ is found during the binary search (e.g., when $|B|$ is empty), then it indicates there does not exist a vector in B that weakly dominates $\vec{b}$. Finally, for `IsDomBySol`, TOA*-Late-BS compares the given vector $\vec{b}$ against the vectors in B_{v_d}, a maximal set of cost-unique non-dominated truncated cost vectors corresponding to labels in $\mathcal{F}(v_d)$, with the same algorithm outlined in Algorithm 8.

To argue that Algorithm 8 is correct, note that every lexicographically sorted list B stores truncated vectors, each of which is of length two. Since the vectors in B are lexicographically sorted, all first components of the vectors in B are monotonically non-decreasing while all second components of the vectors must be monotonically non-increasing. Then, (i) among vectors that are lexicographically larger than $\vec{b}$, none of them can dominate $\vec{b}$; (ii) among vectors that are lexicographically no larger than $\vec{b}$, $\vec{b}^{\max}$ has the smallest second component, and if $b_2^{\max}$ is larger than b_2, $\vec{b}$ cannot be weakly dominated by any existing vector in B. Algorithm 8 has a runtime complexity of $O(\log |B|)$ in the worst case.

Finally, the `FilterAndAddFront` procedure in TOA*-Late-BS is same as the one in EMOA*-Late-BS as aforementioned.

Remark 4. The aforementioned EMOA*-Late-BBST and EMOA*-Late-BS algorithms are two instantiations of EMOA*-Late, and this section presents the instantiation of EMOA*-Early. The BBST-based and BS-based approaches in Sec. 4 and Sec. 5 can be applied to EMOA*-Early to expedite the dominance checking operations. We refer to the resulting algorithms as EMOA*-Early-BBST ($M > 3$), TOA*-Early-BBST ($M = 3$), EMOA*-Early-BS ($M > 3$), and TOA*-Early-BS ($M = 3$) respectively.

Specifically, as discussed in Sec. 3.3, in EMOA*-Early, $\mathcal{F}(v) = \mathcal{F}_{\text{open}}(v) \cup \mathcal{F}_{\text{closed}}(v), v \in V$, and the dimensionality reduction technique can only be applied to $\mathcal{F}_{\text{closed}}(v)$ and is not applicable to the open set $\mathcal{F}_{\text{open}}(u)$. It is because that the first component of the cost vector of labels l that are added to $\mathcal{F}_{\text{open}}(v(l))$ during the search may not be monotonically non-decreasing, and the first component cannot be ignored for dominance checking. We therefore choose to apply the BBST-based or BS-based approaches to $\mathcal{F}_{\text{closed}}(v)$ in these four EMOA*-Early algorithms. In other words, to check if a label l is dominated by any existing labels in $\mathcal{F}(v(l))$, we use the BBST-based or BS-based approaches to check l for dominance against $\mathcal{F}_{\text{closed}}(v(l))$, and then we use a linear scan over $\mathcal{F}_{\text{open}}(v(l))$ to check if l is dominated. Note that all these four EMOA*-Early algorithms are similar to NAMOA*dr with the only difference that the dominance checking related to $\mathcal{F}_{\text{closed}}(v), v \in V$ now uses the BBST-based or BS-based approaches, as opposed to a linear scan over $\mathcal{F}_{\text{closed}}(v)$ as in NAMOA*dr.

6. Experimental results

The main goal of the experimental section is to show the generality and versatility of our framework by implementing different data structure and algorithms. We implement the three procedures `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront` within the EMOA* framework (both EMOA*-Late and EMOA*-Early) using different data structures and approaches, and compare the resulting algorithms against the baseline algorithm NAMOA*dr [19], a state-of-the-art algorithm that can handle an arbitrary number of objectives. All implementations (including the NAMOA*dr baseline) are in the C programming language and use a standard binary heap to implement OPEN.[6]

- Our first implementation of EMOA* is EMOA*-Late-LINEAR, which uses a linked list to represent the front set at each vertex, and as mentioned in Sec. 3.5, the three key procedures `IsDomByFront`, `IsDomBySol` and `FilterAndAddFront` are implemented by running a for-loop over the front set. We refer to this implementation as TOA*-Late-LL (when $M = 3$) and EMOA*-Late-LL ($M > 3$) where 'LL' stands for linked list.
- Our second implementation of EMOA* uses an array (and not a linked list as in TOA*-Late-LL and EMOA*-Late-LL) to represent the front set at each vertex. We refer to this implementation as TOA*-Late-AR (when $M = 3$) and EMOA*-Late-AR ($M > 3$) where 'AR' stands for array.
- The third implementation of EMOA* uses a balanced binary search tree as presented in Sec. 4, where the balanced binary search trees are implemented as AVL-trees using linked lists (i.e., each tree node has two pointers that points to the left and the right child nodes). We refer to this implementation as TOA*-Late-BBST ($M = 3$) and EMOA*-Late-BBST ($M > 3$) where 'BBST' stands for balanced binary search tree.

[6] Our implementation is at https://github.com/carlos-hu70/moaframework.

- The fourth implementation of EMOA* is the binary search-based approach as presented in Sec. 5, where the underlying implementation of each front set is an array, so that the indices of the elements in the array can be used to conduct the binary search. We refer to this implementation as TOA*-Late-BS ($M = 3$) and EMOA*-Late-BS ($M > 3$) where 'BS' stands for binary search.
- The aforementioned four implementations are all based on EMOA*-Late. We also implement the four algorithms mentioned in Sec. 3.3: EMOA*-Early-BBST ($M > 3$), TOA*-Early-BBST ($M = 3$), EMOA*-Early-BS ($M > 3$), and TOA*-Early-BS ($M = 3$).
- Finally, as a baseline we use NAMOA*dr [33]. Here, the front sets (both $\mathcal{F}_{\text{open}}$ and $\mathcal{F}_{\text{closed}}$) at each vertex are implemented by using either a linked list (denoted as NAMOA*dr-LL) or an array (denoted as NAMOA*dr-AR). The original paper [33] implemented NAMOA*dr in Lisp and represented the $\mathcal{F}_{\text{open}}$ and $\mathcal{F}_{\text{closed}}$ using "ordered lists". To the best of our best knowledge, "ordered lists" in Lisp is equivalent to our linked-list implementation in C.

We run tests in the "NY" and "COL" maps, two large-scale city-like road networks, as provided in the dataset of the 9th DIMACS Implementation Challenge, a commonly used dataset for multi-objective planning [11,16,24,30,39].[7] The original maps have, for each edge, two cost components (i.e., two objectives) representing travel distance d and travel time t. Following Casas et al. [40], we use the number of edges q in a path as the third cost component. For the experiment with four and five objectives, we used the economic cost m, which combines toll and fuel consumption according to the road category of New York (NY) (see [19] for additional details), and r, a random integer number between 1 and 100. The order of the objectives are: (q-d-t-m) for four objectives and (q-d-t-m-r) for five objectives.

Additionally, we run tests in the COL map, which is of bigger size than NY. We used three, four and five cost components in this map. Here, we set the edge costs in a different way from the previous tests in the NY map. We use random integer costs for all edges and all cost components. The first cost component c_1 is set to $rnd(1, 1000)$ (where rnd stands for random), a random number between 1 and 1000. The other cost components are generated with the formula $c_i = \rho \times c_1 + \sqrt{1-\rho^2} \times rnd(1, 1000), i = 2, 3 \cdots, M$, where ρ is a parameter that tunes whether a cost component is correlated with c_1. We use $\rho = 0.0001$ such that c_i is uncorrelated with c_1.

All tests are conducted on a Linux machine with 64GB of RAM and a 3.80GHz Intel(R) Core(TM) i7-10700K CPU. We generate instances by randomly sampling start-goal pairs from the graph and report the number of instances in the subsequent tables. We also report the mean, minimum, maximum and median runtime over these instances in the tables. Each instance has a runtime limit of 3600 seconds.

We report experimental results with three, four and five objectives, for the linked-list based implementations (i.e., NAMOA*dr-LL, TOA*-Late-LL, EMOA*-Late-LL, TOA*-Late-BBST and EMOA*-Late-BBST) and for the array-based implementations (i.e., NAMOA*dr-AR, TOA*-Late-AR, EMOA*-Late-AR, TOA*-Late-BS and EMOA*-Late-BS) in the tables. We evaluated 100 instances for three, four and five objectives in NY and COL. In our tests, all implementations time out for some of the instances, which are removed from the tables. This is the reason that the "solved" columns have different total numbers across different tables.

6.1. Experimental results with three objectives

Table 3 shows the number of instances solved within the runtime limit and the runtime statistics for instances with $M = 3$ in NY and COL maps. In the NY map, we can see that TOA*-Early-BBST is faster than NAMOA*dr-LL in terms of the median runtime but slower in terms of the mean runtime, while in the COL map, TOA*-Early-BBST is only faster than NAMOA*dr-LL in terms of the min runtime and slower in all other three runtime metrics. A possible reason is that, with early dominance checking, the front sets are filtered and updated after the generation of each label, which leads to more frequent operations on BBSTs such as rotations and thus slows down the search.

Among the implementations that use the linked-list, TOA*-Late-BBST is the fastest implementation, due to both its late dominance checking and the use of BBSTs to represent the front sets. For the array-based implementations, similar trends to the linked-list based implementations can be observed, and TOA*-Late-BS runs faster than the other three array-based implementations, due to late dominance checking and binary search.

Additionally, in both maps, we observe that the array-based implementations are faster and solve more instances than their linked-list based counterparts. Such a reduction in runtime shows the runtime benefit of using arrays as the underlying implementation of the front set in comparison against using linked-lists. In particular,TOA*-Late-BS runs faster than all implementations, due to the late dominance checking, binary search and the use of arrays. Finally, it is worthwhile noting that, regardless of the data structure (i.e., array or linked-list) or the dominance checking approach (i.e., binary search or linear scan) used by the specific implementation, they all belong to the same EMOA* framework as suggested in this paper, which demonstrates the generality and versatility of the proposed framework.

6.2. Experimental results with four and five objectives

Tables 4 and 5 show the number of instances solved within the runtime limit and the runtime statistics for instances with four and five objectives, respectively. Here, similar trends to the results of $M = 3$ can be observed. Specifically, EMOA*-Late-BS is still the fastest among all the implementations. In addition, as M increases from 3 to 5, the runtime of all implementations increases, and the number of instances solved by each implementation decreases.

[7] This dataset is available at http://www.diag.uniroma1.it/~challenge9/.

Table 3
Instances solved and statistics on runtimes t (in seconds), when an algorithm times out after 3,600 seconds, we use 3,600 in the calculations.

	solved	t_{mean}	t_{max}	t_{min}	t_{median}
	NY with 3 Objectives (avg $\|sols\| = 4,396$) ($\|V\| = 264,346$, $\|E\| = 730,100$)				
NAMOA*dr-LL	98/100	127.97	3,600.00	0.12	1.05
TOA*-Early-BBST	98/100	150.74	3,600.00	0.12	0.88
TOA*-Late-LL	**100/100**	95.17	2,541.85	0.13	0.97
TOA*-Late-BBST	**100/100**	44.23	1,264.57	**0.11**	0.70
NAMOA*dr-AR	**100/100**	47.73	1,732.40	**0.11**	0.59
TOA*-Early-BS	**100/100**	48.50	1,813.65	**0.11**	0.57
TOA*-Late-AR	**100/100**	20.27	670.94	**0.11**	0.54
TOA*-Late-BS	**100/100**	**17.30**	**587.80**	**0.11**	**0.49**
	COL with 3 Objectives (avg $\|sols\| = 6,298$) ($\|V\| = 435,666$, $\|E\| = 1,042,400$)				
NAMOA*dr-LL	**100/100**	54.97	1,280.97	0.21	1.04
TOA*-Early-BBST	**100/100**	70.97	1,014.58	**0.19**	1.36
TOA*-Late-LL	**100/100**	54.52	1,264.80	**0.19**	1.04
TOA*-Late-BBST	**100/100**	34.49	1,017.19	**0.19**	1.37
NAMOA*dr-AR	**100/100**	17.20	227.40	0.20	0.66
TOA*-Early-BS	**100/100**	17.16	216.37	0.20	0.63
TOA*-Late-AR	**100/100**	8.60	136.76	0.20	0.61
TOA*-Late-BS	**100/100**	**7.75**	**99.40**	0.20	**0.60**

Table 4
Instances solved and statistics on runtimes t (in seconds), when an algorithm times out after 3,600 seconds, we use 3,600 in the calculations.

	solved	t_{mean}	t_{max}	t_{min}	t_{median}
	NY with 4 Objectives (avg $\|sols\| = 17,719$)				
NAMOA*dr-LL	38/40	710.46	3,600.00	**0.15**	3.11
EMOA*-Early-BBST	38/40	535.14	3,600.00	**0.15**	3.26
EMOA*-Late-LL	38/40	597.67	3,600.00	**0.15**	2.42
EMOA*-Late-BBST	38/40	560.19	3,600.00	**0.15**	2.04
NAMOA*dr-AR	**40/40**	98.87	1,038.64	**0.15**	1.10
EMOA*-Early-BS	**40/40**	83.21	1,056.59	**0.15**	1.09
EMOA*-Late-AR	**40/40**	108.71	1,171.57	**0.15**	1.02
EMOA*-Late-BS	**40/40**	**71.36**	**929.41**	**0.15**	**0.89**
	COL with 4 Objectives (avg $\|sols\| = 40,500$)				
NAMOA*dr-LL	68/91	1,155.03	3,600.00	0.28	81.45
EMOA*-Early-BBST	68/91	1,139.33	3,600.00	**0.26**	94.04
EMOA*-Late-LL	68/91	1.149.07	3,600.00	0.28	80.26
EMOA*-Late-BBST	69/91	1,100.74	3,600.00	0.27	70.02
NAMOA*dr-AR	86/91	460.16	3,600.00	0.27	12.63
EMOA*-Early-BS	87/91	427.32	3,600.00	0.28	11.93
EMOA*-Late-AR	89/91	367.92	3,600.00	**0.26**	12.15
EMOA*-Late-BS	**91/91**	**253.63**	**3,586.16**	0.28	**10.60**

In addition, we observe that EMOA*-Early sometimes runs faster than their EMOA*-Late counterparts when $M = 5$. For example, in Table 5, EMOA*-Early-BBST runs faster than EMOA*-Late-BBST, and NAMOA*dr-AR runs faster than EMOA*-Late-AR, in terms of both the mean runtime and the median runtime. The reason is that with five objectives, the open list could be huge since it can potentially contain several dominated labels per vertex in the EMOA*-Late based implementations. In contrast, EMOA*-Early based implementations can conduct early dominance checks and prune labels to avoid having many labels in OPEN. This can speed up heap operations when adding labels into or extracting labels from OPEN. In the 59 instances of the COL map, when $M = 3$, the search conducts 9,393 updates of OPEN on average. When $M = 5$, the search conducts 103,723 updates of OPEN on average. This demonstrates that when M is large, not only dominance checking but also heap operations can become important factors that affect the overall computational efficiency of the implementation.

Finally, we observe from Tables 3, 4 and 5 that in certain settings the average runtime of NAMOA*dr-AR is faster than EMOA*-Late-AR. To pinpoint the reason for this, we report in Table 6 the number of dominance checking and filtering operations conducted by both implementations. Noteworthy is that NAMOA*dr-AR and EMOA*-Late-AR use the same implementation of the dominance checking operation. With three objectives NAMOA*dr-AR performs more dominance checking operations than EMOA*-Late-AR (2,310M+1,534M = 3,844M compared to 2,758M) and indeed runs slower. In contrast with four objectives NAMOA*dr-AR performs less dominance checking operations than EMOA*-Late-AR (35,889+1,385M = 37,274M compared to 40,737M) and indeed runs faster.

Table 5
Instances solved and statistics on runtimes t (in seconds), when an algorithm times out after 3,600 seconds, we use 3,600 in the calculations.

	solved	t_{mean}	t_{max}	t_{min}	t_{median}
	NY with 5 Objectives (avg $\|sols\| = 34,142$)				
NAMOA*dr-LL	21/29	1,140.13	3,600.00	0.20	15.62
EMOA*-Early-BBST	21/29	1,108.30	3,600.00	0.19	12.98
EMOA*-Late-LL	21/29	1,139.07	3,600.00	0.20	15.40
EMOA*-Late-BBST	21/29	1,115.68	3,600.00	**0.18**	14.71
NAMOA*dr-AR	26/29	610.63	3,600.00	0.19	2.93
EMOA*-Early-BS	28/29	477.73	3,600.00	**0.18**	2.28
EMOA*-Late-AR	25/29	645.63	3,600.00	0.19	3.60
EMOA*-Late-BS	**29/29**	**444.60**	**3,275.12**	**0.18**	**2.27**
	COL with 5 Objectives (avg $\|sols\| = 31,857$)				
NAMOA*dr-LL	47/59	1,108.73	3,600.00	0.36	422.12
EMOA*-Early-BBST	49/59	1,034.56	3,600.00	**0.33**	400.66
EMOA*-Late-LL	49/59	1,086.85	3,600.00	0.36	418.20
EMOA*-Late-BBST	49/59	1,041.06	3,600.00	**0.33**	444.19
NAMOA*dr-AR	58/59	304.91	3,600.00	**0.33**	45.61
EMOA*-Early-BS	**59/59**	201.01	2,254.37	0.34	35.36
EMOA*-Late-AR	58/59	314.16	3,600.00	**0.33**	58.24
EMOA*-Late-BS	**59/59**	**177.50**	**2,239.07**	0.34	**32.46**

Table 6
Number of dominance checks and filter operations.

	$Check_{\text{Closed}}$	$Check_{\text{Open}}$	$Filter_{\text{Closed}}$	$Filter_{\text{Open}}$
	NY with 3 Cost Components			
NAMOA*dr-AR	2,310M	1,534M	365M	1,447M
TOA*-Late-AR	2,758M		365M	
	NY with 4 Cost Components			
NAMOA*dr-AR	35,889M	1,385M	3,379M	1,273M
EMOA*-Late-AR	40,737M		3,379M	

To understand why NAMOA*dr-AR performs less dominance checking operations than EMOA*-Late-AR, note that in EMOA*-Late-AR, $\mathcal{F}(v)$ is identical to the closed set $\mathcal{F}_{\text{closed}}(v)$. NAMOA*dr-AR further maintains $\mathcal{F}_{\text{open}}(v)$ due to the early dominance checking it performs. EMOA*-Late-AR avoids early checking [11] and converts early checking against $\mathcal{F}_{\text{open}}(v)$ in NAMOA*dr-AR into late checking against $\mathcal{F}_{\text{closed}}(v)$. This conversion can lead to more dominance checking operations while making each checking operation computationally cheaper. However, as the number of objectives increases, the size of the front set increases. In such settings, the dominance checking against $\mathcal{F}_{\text{closed}}(v)$ typically takes more time. NAMOA*dr-AR is able to avoid some of these dominance checking against $\mathcal{F}_{\text{closed}}(v)$ by running early checking against $\mathcal{F}_{\text{open}}(v)$.

7. Conclusion

This article considers the Multi-Objective Shortest-Path Problem (MO-SPP) with an arbitrary number of objectives. We observe that, during the search process of MOA*-like algorithms, the front set at each vertex is computed incrementally by solving the Dominance Checking (DC) problem and Non-Dominated Set Update (NSU) problems iteratively. Based on this observation, we first develop a search framework called Enhanced Multi-Objective A* (EMOA*), which abstracts and highlights the key procedures related to these expensive dominance checking. We show that the existing BOA* algorithm [11] is an instantiation of our EMOA* framework when there are two objectives. Within the EMOA* framework, we develop two different yet closely-related algorithms with fast dominance checking, and discuss their relationship to several other algorithms within the EMOA* framework. Both our algorithms can handle an arbitrary number of objectives. We show that both algorithms are guaranteed to find the exact Pareto-optimal front for MO-SPP. We analyze the runtime complexity of the proposed methods, and verify our framework by implementing and testing several algorithms that follow the EMOA* framework. Our experimental results show that our algorithms runs faster than the baselines on average, and is particularly advantageous for problem instances with three objectives.

For future work, we plan to investigate how new approaches and data structures to solve the NSU problem can be incorporated into our framework. These include (but are not limited to) skip lists [41] and binary space partitioning [23]. These approaches, typically developed by the evolutionary-algorithms community (see, e.g., [42,43] and references within) have the potential to dramatically improve the running time of algorithms instantiated by our framework. In addition, it remains an open question how to analyze the worst-case and average-case number of non-dominated labels at a vertex or over the entire graph after using these fast dominance checking techniques, which may provide new insight about these MOA*-based algorithms. Furthermore, the EARLY and LATE checking strategy spans a spectrum and the middle ground of this spectrum is worthwhile further investigation. Specifically, it is possible for a MOA* search to rely on lightweight early checking, which can be fast but does not remove all dominated labels,

and rely on the late checking as well to take care of the remaining dominated labels when needed. Finally, one can consider further extending the framework to approximate the Pareto-optimal front for MO-SPP as in [30], handling dynamic environments [26,27] or use the algorithms in this article as a building block to solve multi-agent planning problems [44].

CRediT authorship contribution statement

Zhongqiang Ren: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Carlos Hernández:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization. **Maxim Likhachev:** Supervision. **Ariel Felner:** Writing – review & editing. **Sven Koenig:** Writing – review & editing. **Oren Salzman:** Writing – review & editing, Writing – original draft. **Sivakumar Rathinam:** Writing – original draft, Supervision, Funding acquisition. **Howie Choset:** Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grants number 2120219 and 2120529, by the Ministry of Science and Technology, Israel under grants number 3-16079 and 3-17385, and by the United States-Israel Binational Science Foundation (BSF) under grants number 2019703 and 2021643. Carlos Hernández was supported by the National Center for Artificial Intelligence CENIA FB210017, Basal ANID, and the Centro Ciencia & Vida FB210008, Financiamiento Basal ANID. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or any other funding agency.

Data availability

A link to our code is provided in the paper.

References

[1] R.P. Loui, Optimal paths in graphs with stochastic or multidimensional weights, Commun. ACM 26 (1983) 670–676.
[2] B.S. Stewart, C.C. White, Multiobjective A*, J. ACM 38 (1991) 775–814.
[3] L. Mandow, J.L.P. De La Cruz, Multiobjective A* search with consistent heuristics, J. ACM 57 (2008), https://doi.org/10.1145/1754399.1754400.
[4] O. Salzman, A. Felner, C.H. Ulloa, S.-H. Chan, H. Zhang, S. Koenig, Heuristic-search approaches for the multi-objective shortest-path problem: progress and research opportunities, in: International Joint Conference on Artificial Intelligence (IJCAI), 2023.
[5] E. Erkut, S.A. Tjandra, V. Verter, Hazardous materials transportation, in: Handbooks in Operations Research and Management Science, vol. 14, 2007, pp. 539–621.
[6] J. Xu, A. Spielberg, A. Zhao, D. Rus, W. Matusik, Multi-objective graph heuristic search for terrestrial robot design, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2021, pp. 9863–9869.
[7] M. Fu, A. Kuntz, O. Salzman, R. Alterovitz, Toward asymptotically-optimal inspection planning via efficient near-optimal graph search, in: Robotics: Science and Systems, 2019.
[8] M. Fu, O. Salzman, R. Alterovitz, Computationally-efficient roadmap-based inspection planning via incremental lazy search, in: IEEE International Conference on Robotics and Automation, ICRA, 2021, pp. 7449–7456.
[9] J. Montoya, S. Rathinam, Z. Wood, Multiobjective departure runway scheduling using dynamic programming, IEEE Trans. Intell. Transp. Syst. 15 (2013) 399–413.
[10] A. Bronfman, V. Marianov, G. Paredes-Belmar, A. Lüer-Villagra, The maximin hazmat routing problem, Eur. J. Oper. Res. 241 (2015) 15–27.
[11] C. Hernández, W. Yeoh, J.A. Baier, H. Zhang, L. Suazo, S. Koenig, O. Salzman, Simple and efficient bi-objective search algorithms via fast dominance checks, Artif. Intell. 314 (2023) 103807.
[12] P. Serafini, Some considerations about computational complexity for multi objective combinatorial problems, in: Recent Advances and Historical Development of Vector Optimization, Springer, 1987, pp. 222–232.
[13] P. Hansen, Bicriterion path problems, in: Multiple Criteria Decision Making Theory and Application, Springer, 1980, pp. 109–127.
[14] M. Ehrgott, Multicriteria Optimization, vol. 491, Springer Science & Business Media, 2005.
[15] T. Breugem, T. Dollevoet, W. van den Heuvel, Analysis of fptases for the multi-objective shortest path problem, Comput. Oper. Res. 78 (2017) 44–58.
[16] B. Goldin, O. Salzman, Approximate bi-criteria search by efficient representation of subsets of the pareto-optimal frontier, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 31, 2021, pp. 149–158.
[17] S. Ahmadi, G. Tack, D.D. Harabor, P. Kilby, Bi-objective search with bi-directional A*, in: Proceedings of the International Symposium on Combinatorial Search, vol. 12, 2021, pp. 142–144.
[18] H. Zhang, O. Salzman, A. Felner, T.K.S. Kumar, C.H. Ulloa, S. Koenig, Efficient multi-query bi-objective search via contraction hierarchies, in: International Conference on Automated Planning and Scheduling, ICAPS, 2023.
[19] F.-J. Pulido, L. Mandow, J.-L. Pérez-de-la Cruz, Dimensionality reduction in multiobjective shortest path search, Comput. Oper. Res. 64 (2015) 60–70.
[20] H.-T. Kung, F. Luccio, F.P. Preparata, On finding the maxima of a set of vectors, J. ACM 22 (1975) 469–476.
[21] P. Godfrey, R. Shipley, J. Gryz, et al., Maximal vector computation in large data sets, in: VLDB, vol. 5, 2005, pp. 229–240.
[22] P. Godfrey, R. Shipley, J. Gryz, Algorithms and analyses for maximal vector computation, VLDB J. 16 (2007) 5–28.
[23] T. Glasmachers, A fast incremental bsp tree archive for non-dominated points, in: Evolutionary Multi-Criterion Optimization: 9th International Conference, EMO 2017, Münster, Germany, March 19-22, 2017, Proceedings 9, Springer, 2017, pp. 252–266.
[24] Z. Ren, R. Zhan, S. Rathinam, M. Likhachev, H. Choset, Enhanced multi-objective A* using balanced binary search trees, in: Proceedings of the International Symposium on Combinatorial Search, vol. 15, 2022, pp. 162–170.

[25] C.H. Ulloa, H. Zhang, O. Salzman, J. Baier, W. Yeoh, A. Felner, T.K.S. Kumar, S. Koenig, Multi-objective search via lazy and efficient dominance checks, in: International Joint Conference on Artificial Intelligence (IJCAI), 2023.
[26] Z. Ren, S. Rathinam, M. Likhachev, H. Choset, Multi-objective path-based D* lite, IEEE Robot. Autom. Lett. 7 (2022) 3318–3325, https://doi.org/10.1109/LRA.2022.3146918.
[27] Z. Ren, S. Rathinam, M. Likhachev, H. Choset, Multi-objective safe-interval path planning with dynamic obstacles, IEEE Robot. Autom. Lett. 7 (2022) 8154–8161, https://doi.org/10.1109/LRA.2022.3187270.
[28] P. Perny, O. Spanjaard, Near admissible algorithms for multiobjective search, in: ECAI 2008, IOS Press, 2008, pp. 490–494.
[29] A. Warburton, Approximation of pareto optima in multiple-objective, shortest-path problems, Oper. Res. 35 (1987) 70–79.
[30] H. Zhang, O. Salzman, T.K.S. Kumar, A. Felner, C. Hernández Ulloa, S. Koenig, A*pex: efficient approximate multi-objective search on graphs, in: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 32, 2022, pp. 394–403.
[31] H. Zhang, O. Salzman, T.K.S. Kumar, A. Felner, C.H. Ulloa, S. Koenig, Anytime approximate bi-objective search, in: L. Chrpa, A. Saetti (Eds.), International Symposium on Combinatorial Search, SOCS, 2022, pp. 199–207.
[32] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (1968) 100–107.
[33] L. Mandow, J.P. De la Cruz, et al., A new approach to multiobjective A* search, in: IJCAI, vol. 8, Citeseer, 2005.
[34] F. Geißer, P. Haslum, S. Thiébaux, F. Trevizan, Admissible heuristics for multi-objective planning, in: International Conference on Automated Planning and Scheduling, ICAPS, 2022, pp. 100–109.
[35] H. Zhang, O. Salzman, A. Felner, T.K.S. Kumar, S. Skyler, C.H. Ulloa, S. Koenig, Towards effective multi-valued heuristics for bi-objective shortest-path algorithms via differential heuristics, in: International Symposium on Combinatorial Search, SOCS, 2023.
[36] E.Q.V. Martins, On a multicriteria shortest path problem, Eur. J. Oper. Res. 16 (1984) 236–245.
[37] P. Sanders, L. Mandow, Parallel label-setting multi-objective shortest path search, in: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, IEEE, 2013, pp. 215–224.
[38] S. Skyler, S.S. Shperberg, D. Atzmon, A. Felner, O. Salzman, S. Chan, H. Zhang, S. Koenig, W. Yeoh, C.H. Ulloa, Must-expand nodes in multi-objective search [extended abstract], in: International Symposium on Combinatorial Search, SOCS, 2023, pp. 183–184.
[39] V. Kothare, Z. Ren, S. Rathinam, H. Choset, Enhanced multi-objective A* with partial expansion, preprint, arXiv:2212.03712, 2022.
[40] P.M.d.l. Casas, L. Kraus, A. Sedeño-Noda, R. Borndörfer, Multiobjective dijkstra a, preprint, arXiv:2110.10978, 2021.
[41] W. Pugh, Skip lists: a probabilistic alternative to balanced trees, Commun. ACM 33 (1990) 668–676.
[42] J.E. Fieldsend, Data structures for non-dominated sets: implementations and empirical assessment of two decades of advances, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, 2020, pp. 489–497.
[43] Q. Long, X. Wu, C. Wu, Non-dominated sorting methods for multi-objective optimization: review and numerical comparison, J. Ind. Manag. Optim. 17 (2021) 1001–1023.
[44] Z. Ren, S. Rathinam, H. Choset, A conflict-based search framework for multiobjective multiagent path finding, IEEE Trans. Autom. Sci. Eng. 20 (2023) 1262–1274, https://doi.org/10.1109/TASE.2022.3183183.