Multi-objective Search via Lazy and Efficient Dominance Checks

Carlos Hernández[1] , **William Yeoh**[2] , **Jorge A. Baier**[3] , **Ariel Felner**[4] , **Oren Salzman**[5] , **Han Zhang**[6] , **Shao-Hung Chan**[6] , **Sven Koenig**[6]
[1]Universidad San Sebastián, Santiago, Chile
[2]Washington University in St. Louis, Saint Louis, USA
[3]Pontificia Universidad Católica de Chile, Santiago, Chile
[4]Ben-Gurion University of the Negev, Beersheba, Israel
[5]Technion—Israel Institute of Technology, Haifa, Israel
[6]University of Southern California, Los Angeles, USA

Abstract

Multi-objective search can be used to model many real-world problems that require finding Pareto-optimal paths from a specified start state to a specified goal state, while considering different cost metrics such as distance, time, and fuel. The performance of multi-objective search can be improved by making dominance checking—an operation necessary to determine whether or not a path dominates another—more efficient. This was shown in practice by BOA*, a state-of-the-art bi-objective search algorithm, which outperforms previously existing bi-objective search algorithms in part because it adopts a lazy approach towards dominance checking. EMOA*, a recent multi-objective search algorithm, generalizes BOA* to more-than-two objectives using AVL trees for dominance checking. In this paper, we first propose Linear-Time Multi-Objective A* (LTMOA*), a multi-objective search algorithm that implements more efficient dominance checking than EMOA* using simple data structures like arrays. We then propose LazyLT-MOA*, which employs a lazier approach by removing dominance checking during node generation. Our experimental results show that LazyLTMOA* outperforms EMOA* by up to an order of magnitude in terms of runtime.

1 Introduction and Related Work

Multi-objective search can be used to model many real-world problems that involve planning paths with more than one cost metric, such as distance, time, and fuel. Each objective corresponds to minimizing a specific cost metric. Examples of such real-world problems include planning of power-line routes in energy transmission domains [Bachmann *et al.*, 2018], balancing travel distance and risk of exposure when transporting a hazardous material [Bronfman *et al.*, 2015], and inspecting a region of interest using cameras placed on-board robotic platforms [Fu *et al.*, 2019; Fu *et al.*, 2021]. While single-objective search produces shortest paths with respect to a single cost metric, it needs to be efficiently generalized to multi-objective search for the aforementioned real-world problems. Generally, the desired output of multi-objective search is a set of non-dominated paths with respect to the different cost metrics.

More formally, in multi-objective search, we are given a directed graph with multiple costs annotating each edge, a specified start state, and a specified goal state. A path π is considered to be *better than*, i.e., to *dominate*, another path π' if and only if π is not worse than π' on any cost metric and π is better than π' on at least one cost metric, and a *Pareto-optimal solution* is a path from the start state to the goal state that are not dominated by *any* path from the start state to the goal state. We are interested in computing the Pareto-optimal solution set, that is, the set of all Pareto-optimal solutions.

Importantly, different variants of the problem exist such as approximating the Pareto-optimal solution set [Salzman and Goldin, 2021; Zhang *et al.*, 2022b; Pangilinan and Janssens, 2007; Li *et al.*, 2015] and computing the Pareto-optimal solution set in an anytime manner [Zhang *et al.*, 2022a]. Researchers have also computed a single solution in the Pareto-optimal solution set given bounds on its cost [Skyler *et al.*, 2022] and using bi-directional search [Ahmadi *et al.*, 2021]. However, these are out of the scope of the paper.

Existing algorithms for computing the Pareto-optimal solution set include NAMOA* [Mandow *et al.*, 2005; Mandow and De La Cruz, 2010], NAMOA* with dimensionality reduction (NAMOA*dr) [Pulido *et al.*, 2015], Enhanced Multi-Objective A* (EMOA*) [Ren *et al.*, 2022], and Bi-Objective A* (BOA*) [Hernández *et al.*, 2023]. All of these algorithms perform *dominance checks* to determine if a path from the start state to a specific state has the potential to be extended to a Pareto-optimal solution. However, they differ in how the dominance checks are implemented and interleaved with the search. Both NAMOA* and NAMOA*dr can be used for any number of objectives and perform dominance checks eagerly: Every time they generate a path to a specific state, they *eagerly* check if this path is dominated by any previously generated path to the same state, which involves time-consuming operations that iterate over generated paths. BOA* shows that some of these time-consuming operations can be alleviated using a lazy approach for dominance checks and performs all dominance checks in constant time by doing so. However, it is restricted to solving problems with only two objectives. The recently proposed EMOA* algorithm generalizes the lazy dominance checks of BOA* to more than two objective. For problems with three objectives, it uses AVL trees and performs dominance checks in logarithmic time when generating a path.

In this paper, we first investigate different data structures that can be used for dominance checks and propose Linear-Time Multi-Objective A* (LTMOA*), an multi-objective search algorithm that implements a more efficient dominance checking than EMOA* using simple data structures like arrays. We then propose an even "lazier" approach towards dominance checking than those of EMOA* and LTMOA*. The resulting algorithm, LazyLTMOA*, distinguishes from EMOA* and LTMOA* by removing dominance checking during node generation and only performing dominance checking when a path is selected for expanding.

We evaluate LTMOA* and LazyLTMOA* on road network instances with up to five objectives. Empirically, we show that LazyLTMOA* is about 60% faster than LTMOA* due to less dominance checking. With our efficient dominance checking approach, LazyLTMOA* substantially outperforms EMOA* by up to an order of magnitude in terms of runtime.

2 Notation and Problem Definition

A multi-objective *search graph* is a tuple $(S, E, \mathbf{c})$, where S is the finite set of *states*, $E \subseteq S \times S$ is the finite set of edges, and $\mathbf{c} : E \to (\mathbb{R}_{\geq 0})^n$ is a *cost function* that associates an n-tuple of non-negative real costs with each edge, where n is the number of components. $Succ(s) = \{t \in S \mid (s,t) \in E\}$ denotes the successors of state s. A *path* π from s_1 to s_m is a sequence of states $s_1, s_2, \ldots, s_m$ such that $(s_i, s_{i+1}) \in E$ for all $i \in \{1, \ldots, m-1\}$.

Boldface font is used to represent n-tuples. Unless defined otherwise, we assume a n-tuple $\mathbf{v}$ has the form $\mathbf{v} = (v_1, v_2, \ldots, v_n)$, thus v_i denotes the i-th component of an n-tuple. The addition of two n-tuples $\mathbf{u}$ and $\mathbf{v}$ and the multiplication of a real-valued scalar k and a tuple $\mathbf{u}$ are defined in the natural way; that is, $\mathbf{u} + \mathbf{v} = (u_1 + v_1, \ldots, u_n + v_n)$ and $k\mathbf{u} = (ku_1, \ldots, ku_n)$. $\mathbf{c}(\pi) = \sum_{i=1}^{m-1} \mathbf{c}(s_i, s_{i+1})$ is the cost of path $\pi = s_1, \ldots, s_m$.

Given two n-tuples $\mathbf{u}$ and $\mathbf{v}$, we say that $\mathbf{u}$ *weakly dominates* $\mathbf{v}$, denoted as $\mathbf{u} \preceq \mathbf{v}$, if $v_i \leq u_i$ holds for every $i \in \{1, \ldots, n\}$. We say $\mathbf{u}$ *dominates* $\mathbf{v}$, denoted as $\mathbf{u} \prec \mathbf{v}$, if $\mathbf{u} \preceq \mathbf{v}$ and $\mathbf{u} \neq \mathbf{v}$ hold. We say that path π *dominates* (resp. *weakly dominates*) path π', denoted as $\pi \prec \pi'$ (resp. $\pi \preceq \pi'$) if $\mathbf{c}(\pi) \prec \mathbf{c}(\pi')$ (resp. $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi')$).

A search instance is defined as a tuple $P = (S, E, \mathbf{c}, s_{\text{start}}, s_{\text{goal}})$, where $(S, E, \mathbf{c})$ is a search graph and $s_{\text{start}}, s_{\text{goal}} \in S$ are the *start* and *goal* states, respectively. Given a search instance P, a *Pareto-optimal solution set to s_{goal}* (from s_{start}), denoted as $sols(s_{\text{goal}})$, contains every path π from s_{start} to s_{goal} with the property that, for every other path π' from s_{start} to s_{goal}, $\pi' \not\prec \pi$ holds; that is, $sols(s_{\text{goal}})$ contains all non-dominated paths from s_{start} to s_{goal}. In this paper, we aim to find any maximal subset of the Pareto-optimal solution set such that each solution in the subset has a unique cost of path. We refer to this subset as a *cost-unique Pareto-optimal solution set*.

A heuristic function $\mathbf{h}$ maps each state s in S to a n-tuple in $(\mathbb{R}_{\geq 0})^n$ domain that estimates the cost of a path from state s to s_{goal}. Given a heuristic function $\mathbf{h}$, the $\mathbf{h}$-value of state s is denoted as $\mathbf{h}(s)$. $\mathbf{h}$ is *admissible* iff $\mathbf{h}(s) \preceq \mathbf{c}(\pi)$ holds for every state s and every path π that traverses s from s_{start} to s_{goal}. Similarly, $\mathbf{h}$ is consistent iff (i) $\mathbf{h}(s_{\text{goal}}) = \mathbf{0}$ and (ii) $\mathbf{h}(s) \preceq \mathbf{c}(s,t) + \mathbf{h}(t)$ for every $(s,t) \in E$. That is, all components of $\mathbf{h}$ are consistent for the corresponding components of the cost function.

Algorithm 1: NAMOA*

```
Input  : A search problem $(S, E, \mathbf{c}, s_{\text{start}}, s_{\text{goal}})$ and a consistent
         heuristic function $\mathbf{h}$
Output: A cost-unique Pareto-optimal solution set
1  $sols \leftarrow \emptyset$
2  for each $s \in S$ do
3      $\mathbf{G}_{\text{op}}(s) \leftarrow \emptyset$; $\mathbf{G}_{\text{cl}}(s) \leftarrow \emptyset$
4  $x \leftarrow$ new node with $s(x) = s_{\text{start}}$
5  $\mathbf{g}(x) \leftarrow$ zero in all dimensions
6  Add $\mathbf{g}(x)$ to $\mathbf{G}_{\text{op}}(s(x))$
7  $p(x) \leftarrow null$
8  $\mathbf{f}(x) \leftarrow \mathbf{h}(s_{\text{start}})$
9  Initialize $Open$ and add $x$ to it
10 while $Open \neq \emptyset$ do
11     Extract a node $x$ from $Open$ with the lexicographically
         smallest $f$-value
12     Remove $\mathbf{g}(x)$ from $\mathbf{G}_{\text{op}}(s(x))$
13     if IsDominated($\mathbf{f}(x)$, $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$) then
14         continue
15     Add $\mathbf{g}(x)$ to $\mathbf{G}_{\text{cl}}(s(x))$
16     if $s(x) = s_{goal}$ then
17         Add $x$ to $sols$
18         continue
19     for each $t \in Succ(s(x))$ do
20         $x' \leftarrow$ new node with $s(x') = t$
21         $\mathbf{g}(x') \leftarrow \mathbf{g}(x) + \mathbf{c}(s(x), t)$
22         $p(x) \leftarrow x$
23         $\mathbf{f}(x') \leftarrow \mathbf{g}(x') + \mathbf{h}(t)$
24         if IsDominated($\mathbf{g}(x')$, $\mathbf{G}_{\text{cl}}(t) \cup \mathbf{G}_{\text{op}}(t)$) or
             IsDominated($\mathbf{f}(x')$, $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$) then
25             continue
26         UpdateOpen($x'$)
27 return $sols$
```

Algorithm 2: IsDominated

```
Input  : A vector $\mathbf{v}$ and a set of vector $\mathbf{V}$
Output: $true$ or $false$
1  for each $\mathbf{v}' \in \mathbf{V}$ do
2      if $\mathbf{v}' \preceq \mathbf{v}$ then
3          return true
4  return false
```

3 Algorithmic Background

We now describe the multi-objective search algorithms that form the building blocks for our new algorithm. All algorithms perform a best-first search and compute a cost-unique Pareto-optimal solution set for a given search instance. Thus, we start with general notations and assumptions that will be used for all algorithms (including our own) and then describe each algorithm while highlighting the differences. All the al-

Algorithm 3: UpdateOpen

```
Input  : A node $x$
Output: $Open$ and $\mathbf{G}_{\text{op}}(s(x))$ updated
1 for each $\mathbf{g}' \in \mathbf{G}_{op}(s(x))$ do
2     if $\mathbf{g}(x) \preceq \mathbf{g}'$ then
3         Remove $\mathbf{g}'$ from $\mathbf{G}_{\text{op}}(s(x))$
4         Remove nodes corresponding to $\mathbf{g}'$ from $Open$
5 Add $\mathbf{g}(x)$ to $\mathbf{G}_{\text{op}}(s(x))$
6 Add $x$ to $Open$
7 return
```

gorithms maintain an $Open$ list, containing the frontier of the search tree (i.e., the generated but not-yet-expanded nodes) and a set of solutions $sols$. Here, a node x is associated with a state $s(x) \in S$, a $\mathbf{g}$-value $\mathbf{g}(x)$, and a parent node $p(x)$. We also define $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}(s(x))$ as the $\mathbf{f}$-value of x. Conceptually, x corresponds to a path from s_{start} to $s(x)$ with cost $\mathbf{g}(x)$. The path can be constructed by backtracking along the parent nodes. We assume that the reader is familiar with the properties of A* when used with consistent $\mathbf{h}$-values, for example, that the sequence of expanded nodes has monotonically non-decreasing $\mathbf{f}$-values.

NAMOA*: For any state s, NAMOA* (Algorithm 1) maintains two sets of $\mathbf{g}$-values, $\mathbf{G}_{\text{cl}}(s)$ and $\mathbf{G}_{\text{op}}(s)$.[1] These sets store the non-dominated $\mathbf{g}$-values for all the expanded nodes and for all the generated but not-yet-expanded nodes associated with state s, respectively. These two sets are used to perform dominance checks in order to prune nodes that cannot be part of the Pareto-optimal set. To check if a given n-tuple $\mathbf{v}$ is dominated by a n-tuple in a given set of n-tuples $\mathbf{V}$, the algorithm iterates through $\mathbf{V}$, which takes linear time in $|\mathbf{V}|$, as shown in Algorithm 2.

In each iteration, NAMOA* extracts a node x from $Open$ whose $\mathbf{f}$-value is not dominated by the $\mathbf{f}$-value of any node in $Open$. One way to select such a non-dominated node is to select a node with the lexicographically smallest $\mathbf{f}$-value (Line 11). Node x is pruned if $\mathbf{f}(x)$ is dominated by a $\mathbf{g}$-value in $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$ (Lines 13-14). Otherwise, NAMOA* adds $\mathbf{g}(x)$ to $\mathbf{G}_{\text{cl}}(s(x))$ (Line 15) and expands x. If state $s(x)$ is the goal state s_{goal}, then NAMOA* adds (the corresponding path of) x to $sols$ (Lines 16-18). Otherwise, it generates a child node x' for each successor t of state $s(x)$. Node x' is pruned if its $\mathbf{g}$-value is dominated by a $\mathbf{g}$-value in $\mathbf{G}_{\text{cl}}(t) \cup \mathbf{G}_{\text{op}}(t)$ or if its $\mathbf{f}$-value is dominated by a $\mathbf{g}$-value in $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$ (Lines 24-25). If node x' is not pruned, then NAMOA* removes all nodes with state t and whose $\mathbf{g}$-values are dominated by $\mathbf{g}(x')$ from $Open$ and their $\mathbf{g}$-values from $\mathbf{G}_{\text{op}}(t)$, and adds node x' to $Open$ and its $\mathbf{g}$-value to $\mathbf{G}_{\text{op}}(s(x'))$ (see Algorithm 3).

NAMOA*dr: Recall that each dominance check performed by NAMOA* (Lines 13 and 24, Algorithm 1) requires a linear-time iteration over $\mathbf{G}_{\text{cl}}(s)$ and $\mathbf{G}_{\text{op}}(s)$ for each state s considered. This operation is time-consuming and can often dominate the algorithm's runtime. NAMOA*dr [Pulido *et al.*, 2015] improves the processes by using a technique called *dimensionality reduction* to reduce the size of $\mathbf{G}_{\text{cl}}$ and thus reduce the time taken to perform dominance checks subsequently.

[1] 'cl' and 'op' are shorthand for 'closed' and 'open', respectively.

NAMOA*dr uses a consistent heuristic and extracts a node with the lexicographically smallest $\mathbf{f}$-value among all nodes in $Open$ in each iteration. Thus, when NAMOA*dr extracts or generates a node x, we always have that $g_1(x) \geq \max\{g_1 \mid \mathbf{g} \in \mathbf{G}_{\text{cl}}(s(x))\}$ and that $f_1(x) \geq \max\{g_1 \mid \mathbf{g} \in \mathbf{G}_{\text{cl}}(s_{\text{goal}})\}$. Therefore, *there is no need to consider the first component of* $\mathbf{g}$- *or* $\mathbf{f}$ *in dominance checks*. To this end, let the *truncated vector* of a vector $\mathbf{v}$ with length n, denoted as $\text{Tr}(\mathbf{v})$, be the vector consisting of its last $n-1$ elements. That is, $\text{Tr}(\mathbf{v}) = (v_2, \ldots, v_n)$. For a state s, NAMOA*dr only needs to maintain the set of non-dominated truncated $\mathbf{g}$-values for all expanded nodes, namely $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s)$, which is often significantly smaller than the size of $\mathbf{G}_{\text{cl}}(s)$ [Pulido *et al.*, 2015]. As the $\mathbf{g}$-values of generated nodes are not necessarily monotonically and lexicographically increasing, to check whether a generated node x is dominated by some node in $Open$ associated with state $s(x)$, NAMOA*dr still needs to completely iterate over $\mathbf{G}_{\text{op}}(s(x))$.

BOA*: BOA* [Hernández *et al.*, 2023] is a bi-objective (that is, with exactly two objectives) search algorithm which also builds upon the dimensionality reduction technique of NAMOA*dr. In the specific setting of bi-objective search, the truncated $\mathbf{g}$-value is a single number g_2 (2 is the index of the 2nd and last dimension), and a non-dominated truncated $\mathbf{g}$-value set can be represented by the minimum g_2-value of all $\mathbf{g}$-values. For all expanded nodes associated with some state s, BOA* stores this minimum value in $g_2^{\min}(s)$. When a node x is extracted from $Open$ or generated, BOA* prunes node x if $\mathbf{f}(x) \geq g_2^{\min}(s_{\text{goal}})$ or if $\mathbf{g}(x) \geq g_2^{\min}(s(x))$ holds.

BOA* and NAMOA*dr differ in the way dominance checks are performed. BOA* performs a dominance check for a node x (both when generating it and when extracting it from $Open$) by matching x against $g_2^{\min}(s(x))$ (which is equivalent to performing dominance check in $Closed$). But when x is generated, BOA* does not check if the node is dominated by some node in $Open$. Therefore, the greatest advantage of BOA* is that all dominance checks of BOA* are done in constant time (against $g_2^{\min}$). But since no pruning is done with nodes in $Open$, there is a risk in BOA* of having a larger $Open$ compared to NAMOA*dr. Nevertheless, Hernández *et al.* [2023] showed empirically that BOA* significantly outperforms both NAMOA* and NAMOA*dr in terms of runtime (due to the much smaller dominance check overhead) on a wide suite of benchmarks of road network instances.

Multi-Objective Search Framework and EMOA*: Recently, Ren *et al.* [2022] proposed a Multi-Objective Search Framework (MOSF) that generalizes BOA* to more than two objectives. Similar to BOA*, when a node x is generated, a particular algorithm in MOSF performs dominance checks against only $\mathbf{G}_{\text{cl}}(s(x))$ and $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$ while ignoring $Open$. The EMOA* [Ren *et al.*, 2022] algorithm is a particular case of MOSF. EMOA* also utilizes the dimensionality reduction technique. It extracts nodes in the increasing lexicographic order and maintains the set of non-dominated truncated $\mathbf{g}$-values $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s)$ for each state s. Specifically, EMOA* uses an AVL tree to store $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s)$ for each state s. With three objectives, the dominance check against $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s)$ takes in logarithmic time in terms of the number of elements in $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s)$,

Algorithm 4: `RemoveDominated`

```
Input  : A vector $\mathbf{v}$ and a set of vectors $\mathbf{V}$
Output: $\mathbf{V}$ updated
1  for each $\mathbf{v}' \in \mathbf{V}$ do
2      if $\mathbf{v} \preceq \mathbf{v}'$ then
3          Remove $\mathbf{v}'$ from $\mathbf{V}$
4  return
```

Algorithm 5: LTMOA*

```
Input  : A search problem $(S, E, \mathbf{c}, s_{\text{start}}, s_{\text{goal}})$ and a consistent
         heuristic function $\mathbf{h}$
Output: A cost-unique Pareto-optimal solution set
1  $sols \leftarrow \emptyset$
2  for each $s \in S$ do
3      $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s) \leftarrow \emptyset$
4  $x \leftarrow$ new node with $s(x) = s_{\text{start}}$
5  $\mathbf{g}(x) \leftarrow$ zero in all dimensions
6  $p(x) \leftarrow null$
7  $\mathbf{f}(x) \leftarrow \mathbf{h}(s_{\text{start}})$
8  Initialize $Open$ and add $x$ to it
9  while $Open \neq \emptyset$ do
10     Extract a node $x$ from $Open$ with the lexicographically
         smallest $f$-value
11     if IsDominated($\text{Tr}(\mathbf{g}(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$) or
         IsDominated($\text{Tr}(\mathbf{f}(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$) then
12         continue
13     RemoveDominated($\text{Tr}(\mathbf{g}(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$)
14     Add $\mathbf{g_T}(x)$ to $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$
15     if $s(x) = s_{goal}$ then
16         Add $x$ to $sols$
17         continue
18     for each $t \in Succ(s(x))$ do
19         $y \leftarrow$ new node with $s(y) = t$
20         $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{c}(s(x), t)$
21         $p(y) \leftarrow x$
22         $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$
23         if IsDominated($\text{Tr}(\mathbf{g}(y)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(t)$) or
             IsDominated($\text{Tr}(\mathbf{f}(y)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$) then
24             continue
25         Add $y$ to $Open$
26 return $sols$
```

denoted as $|\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)|$. However, with more than three objectives, EMOA*'s dominance checks still takes linear time on $|\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)|$. Ren *et al.* [2022] showed empirically that EMOA* outperforms NAMOA*dr in terms of runtime on road network instances with three objectives.

4 LTMOA*

In this section, we describe Linear Time MOA* (LTMOA*), a new particular case of MOSF that performs dominance check in linear time using a linked list or an array to store $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$ for each state s. Algorithm 5 shows the pseudo-code for LTMOA*. LTMOA* maintains $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$, a set of non-dominated truncated $\mathbf{g}$-values for each state s. In contrast to NAMOA* and NAMOA*dr, LTMOA* does not maintain $\mathbf{G}_{\text{op}}(s)$. When extracting or generating a node x, LTMOA* prunes x if (*i*) $\text{Tr}(\mathbf{f}(x))$ is weakly dominated by any vectors in $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$; or if (*ii*) $\text{Tr}(\mathbf{g}(x))$ is weakly dominated by any vector in $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$ (Lines 11 and 23). Similar to BOA*, LTMOA* does not check if a generated node is dominated by

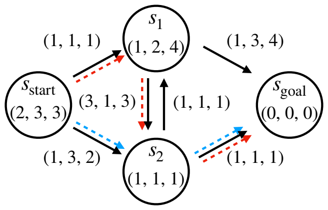

Figure 1: An example search instance with three objectives. The vector inside each state is its $\mathbf{h}$-value.

Iter	*Open* $\langle s(x), \mathbf{g}(x), \mathbf{f}(x)\rangle$	Update of $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$
1	$\langle s_{\text{start}}, (0,0,0), (2,3,3)\rangle*$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{start}}) = \{(0,0)\}$
2	$\langle s_1, (1,1,1), (2,3,5)\rangle*$ $\langle s_2, (1,3,2), (2,4,3)\rangle$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_1) = \{(1,1)\}$
3	$\langle s_2, (1,3,2), (2,4,3)\rangle*$ $\langle s_2, (4,2,4), (5,3,5)\rangle$ $\langle s_{\text{goal}}, (2,4,5), (2,4,5)\rangle$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_2) = \{(3,2)\}$
4	$\langle s_{\text{goal}}, (2,4,3), (2,4,3)\rangle*$ $\langle s_2, (4,2,4), (5,3,5)\rangle$ $\langle s_{\text{goal}}, (2,4,5), (2,4,5)\rangle$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}}) = \{(4,3)\}$
5	$\langle s_2, (4,2,4), (5,3,5)\rangle$ $\langle s_{\text{goal}}, (2,4,5), (2,4,5)\rangle*$	
6	$\langle s_2, (4,2,4), (5,3,5)\rangle*$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_2) = \{(3,2),(2,4)\}$
7	$\langle s_{\text{goal}}, (5,3,5), (5,3,5)\rangle*$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}}) = \{(4,3),(3,5)\}$
8	empty	

Table 1: Trace of *Open* and $\mathbf{G}_{\text{cl}}^{\text{Tr}}$ in each iteration of LTMOA* on solving the example search instance. "$*$" marks the node that is extracted in that iteration.

nodes in *Open* associated with the same state. Thus, multiple copies of the same state (one dominating the other) may be simultaneously in *Open*. Then, after choosing node x for expansion, LTMOA* removes all (truncated) $\mathbf{g}$-values that are dominated by $\text{Tr}(\mathbf{g}(x))$ from $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$ (Line 13) and then inserts $\text{Tr}(\mathbf{g}(x))$ to $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$ (Line 14).

In Algorithm 5, we highlight some differences of LTMOA* compared to NAMOA*dr in blue. When generating a node x, NAMOA*dr performs more dominance checks than LTMOA* as it needs to iterate over $\mathbf{G}_{\text{op}}(s(x))$. More specifically, assuming that x is not pruned, NAMOA*dr iterates over $\mathbf{G}_{\text{op}}(s(x))$ twice: It first checks if $\mathbf{g}(x)$ is dominated by any $\mathbf{g}$-value in $\mathbf{G}_{\text{op}}(s(x))$ (Line 24, Algorithm 1), and then it needs to check if each $\mathbf{g}$-value in $\mathbf{G}_{\text{op}}(s(x))$ should be removed. In contrast, LTMOA* does not perform the dominance checks with $\mathbf{G}_{\text{op}}(s(x))$ but, instead, performs more dominance checks when it extracts x for expansion.

The main difference between LTMOA* and EMOA* is how dominance checks are implemented. In EMOA* the dominance checks are implemented with AVL trees. We provide two other approaches for dominance checks, which use different data structures than AVL trees: (1) Linked List: For each state s, $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$ is stored as a linked list. Both `IsDominated` and `RemoveDominated` iterate through the entire linked list and hence take linear time of $|\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)|$. (2) Array: For each state s, $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$ is stored as an array. When removing a $\mathbf{g}$-value from $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$, LTMOA* first swap this $\mathbf{g}$-value with the last $\mathbf{g}$-value in the array and then pop the last $\mathbf{g}$-value from the array, which takes only constant time.

Figure 1 shows a toy problem instance with three objectives with Pareto-optimal paths in this problem instance:

$s_{\text{start}}, s_2, s_{\text{goal}}$ (shown by blue arrows) with cost $(2, 4, 3)$ and $s_{\text{start}}, s_1, s_2, s_{\text{goal}}$ (shown by red arrows) with cost $(5, 3, 5)$. Table 1 shows a trace of *Open* and changes to $\mathbf{G}^{\text{Tr}}_{\text{cl}}$ in each iteration of LTMOA*. In Iterations 1-2, LTMOA* expands nodes $\langle s_{\text{start}}, (0, 0, 0), (2, 3, 3)\rangle$ and $\langle s_1, (1, 1, 1), (2, 3, 5)\rangle$, and then generates node $\langle s_{\text{goal}}, (2, 4, 5), (2, 4, 5)\rangle$, which is associated with s_{goal}. However, in Iterations 3-4, LTMOA* finds a solution with cost $(2, 4, 3)$. Hence, node $\langle s_{\text{goal}}, (2, 4, 5), (2, 4, 5)\rangle$ is extracted and pruned in Iteration 5. In Iteration 3, node $\langle s_1, (2, 4, 3), (3, 6, 7)\rangle$ is also generated when expanding node $\langle s_2, (1, 3, 2), (2, 4, 3)\rangle$. However, it is not added to *Open* since it is pruned (and hence not shown in the table) as $\text{Tr}(2, 4, 3) = (4, 3)$ is dominated by $(1, 1) \in \mathbf{G}^{\text{Tr}}_{\text{cl}}(s_1)$. In Iteration 6, the node associated with s_1, generated when expanding node $\langle s_2, (4, 2, 4), (5, 3, 5)\rangle$ is also pruned.

If we use NAMOA*dr to solve this problem instance, node $\langle s_{\text{goal}}, (2, 4, 5), (2, 4, 5)\rangle$ will be removed from *Open* in Iteration 3 when node $\langle s_{\text{goal}}, (2, 4, 3), (2, 4, 3)\rangle$ is inserted to *Open*, and the algorithm will terminate in Iteration 7. Despite having fewer iterations, NAMOA*dr needs to perform more dominance checks: In Iteration 2, when generating node $\langle s_2, (4, 2, 4), (5, 3, 5)\rangle$, NAMOA*dr needs to check if its $\mathbf{g}$-value $(4, 2, 4)$ and $(1, 3, 2)$, that is, the $\mathbf{g}$-value in $\mathbf{G}_{\text{op}}(s_2)$, dominates each other. Meanwhile LTMOA* inserts node $\langle s_2, (4, 2, 4), (5, 2, 5)\rangle$ into *Open* without performing such dominance checks, and it only needs to perform one additional dominance check between $(2, 4)$ and $(3, 2)$ in Iteration 6 before expanding the node.

5 A Lazy Multi-Objective Search Framework

We propose an even "lazier" framework in terms of dominance checks, Lazy MOSF, which is a simple extension of MOSF where dominance checks during node generation is totally removed. To explain how the framework works, we describe LazyLTMOA*, a Lazy-MOSF variant of LTMOA*. While LTMOA* performs dominance checks against truncated $\mathbf{g}$-values of expanded nodes when a node is generated or extracted from *Open*, LazyLTMOA* performs dominance checks only when a node is extracted from *Open* (line 11 of Algorithm 5). We omit the pseudo code for LazyLTMOA* since if can be obtained by removing Lines 23-24 from Algorithm 5.

Consider LazyLTMOA* and the problem instance in Figure 1. Table 2 shows a trace of *Open* and changes to $\mathbf{G}^{\text{Tr}}_{\text{cl}}$ in each iteration of LazyLTMOA*. In Iteration 3, node $\langle s_1, (2, 4, 3), (3, 6, 7)\rangle$ is generated by LazyLTMOA* when expanding $\langle s_2, (1, 3, 2), (2, 4, 3)\rangle$ and inserted into *Open* (whereas LTMOA* prunes $\langle s_1, (2, 4, 3), (3, 6, 7)\rangle$). Later, in Iteration 6, node $\langle s_1, (2, 4, 3), (3, 6, 7)\rangle$ is extracted from *Open* and then pruned. Similarly, in Iteration 7, node $\langle s_1, (5, 3, 5), (6, 5, 9)\rangle$ is also generated and inserted into *Open* when expanding $\langle s_2, (4, 2, 4), (5, 3, 5)\rangle$. Although LazyLTMOA* has more iterations than LTMOA*, it performs fewer dominance checks.

Table **??** summarizes the properties of different algorithms, including the number of objectives that they are designed for (second column); the dominance checks that are done when a node x is generated and when it is extracted from *Open* (third and fourth columns, respectively); whether they exploit dimensionality reduction (fifth column); whether they avoid dominance checks in *Open*; and weathesr they avoid dominance checks when a node is generated (last column). When comparing LTMOA* or EMOA* to NAMOA*dr, we observe the following: NAMOA*dr performs dominance checks in *Open* but it only iterates over $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_{\text{goal}})$ when extracting a node for expansion. LTMOA* and EMOA* avoid dominance checks in *Open* but it needs to iterate over both $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s(x))$ and $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_{\text{goal}})$ when extracting a node. Therefore, LTMOA* and EMOA* performs fewer dominance checks for x if $|\mathbf{G}^{\text{Tr}}_{\text{cl}}(s(x))| < |\mathbf{G}_{\text{op}}(s(x))|$. LazyLTMOA* only iterates $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_{\text{goal}})$ and $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s(x))$ when extracting a node from *Open* and hence performs the fewest dominance checks.

Iter	*Open* $\langle s(x), \mathbf{g}(x), \mathbf{f}(x)\rangle$	Update of $\mathbf{G}^{\text{Tr}}_{\text{cl}}(s(x))$
1	$\langle s_{\text{start}}, (0,0,0), (2,3,3)\rangle *$	$\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_{\text{start}}) = \{(0,0)\}$
2	$\langle s_1, (1,1,1), (2,3,5)\rangle *$ $\langle s_2, (1,3,2), (2,4,3)\rangle$	$\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_1) = \{(1,1)\}$
3	$\langle s_2, (1,3,2), (2,4,3)\rangle *$ $\langle s_2, (4,2,4), (5,3,5)\rangle$ $\langle s_{\text{goal}}, (2,4,5), (2,4,5)\rangle$	$\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_2) = \{(3,2)\}$
4	$\langle s_{\text{goal}}, (2,4,3), (2,4,3)\rangle *$ $\langle s_1, (2,4,3), (3,6,7)\rangle$ $\langle s_2, (4,2,4), (5,3,5)\rangle$ $\langle s_{\text{goal}}, (2,4,5), (2,4,5)\rangle$	$\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_{\text{goal}}) = \{(4,3)\}$
5	$\langle s_1, (2,4,3), (3,6,7)\rangle$ $\langle s_2, (4,2,4), (5,3,5)\rangle$ $\langle s_{\text{goal}}, (2,4,5), (2,4,5)\rangle *$	
6	$\langle s_1, (2,4,3), (3,6,7)\rangle *$ $\langle s_2, (4,2,4), (5,3,5)\rangle$	
7	$\langle s_2, (4,2,4), (5,3,5)\rangle *$	$\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_2) = \{(3,2),(2,4)\}$
8	$\langle s_{\text{goal}}, (5,3,5), (5,3,5)\rangle *$ $\langle s_1, (5,3,5), (6,5,9)\rangle$	$\mathbf{G}^{\text{Tr}}_{\text{cl}}(s_{\text{goal}}) = \{(4,3),(3,5)\}$
9	$\langle s_1, (5,3,5), (6,5,9)\rangle *$	
10	empty	

Table 2: Trace of *Open* and $\mathbf{G}^{\text{Tr}}_{\text{cl}}$ in each iteration of LazyLTMOA* on solving the example search instance. "$*$" marks the node that is extracted in that iteration.

6 Theoretical Results

We now prove that LTMOA* returns a cost-unique Pareto-optimal solution set (Theorem 1) using the general proof strategy for BOA*. The same property can be proved for LazyLTMOA* (Theorem 2) following the same proof. As the proofs for some lemmas are straightforward generalizations of those for BOA*, we also omit them here for brevity. Instead, we focus on those with significant differences compared to the ones for BOA* as a result of generalizing from two to more objectives.

Lemma 1. *For each generated (or about to be generated but pruned) node x with its parent node p, it holds that $f_1(x) \geq f_1(p)$ and $\text{Tr}(\mathbf{f}(x)) \succeq \text{Tr}(\mathbf{f}(p))$.*

Lemma 2. *The sequences of extracted nodes and of expanded nodes have monotonically non-decreasing f_1-values.*

Lemma 3. *Two nodes x_1 and x_2 with the same state s that are sequentially expanded will satisfy $\text{Tr}(\mathbf{f}(x_1)) \not\preceq \text{Tr}(\mathbf{f}(x_2))$.*

Proof Sketch: Assume for a proof by contradiction that LTMOA* expands node x_1 with state s before node x_2 with state s, that it expands no node with state s after node x_1 and before node x_2, and that $\text{Tr}(\mathbf{f}(x_1)) \preceq \text{Tr}(\mathbf{f}(x_2))$. Then,

Algorithm	#Obj.	Node Generation	Node Extraction	DR	AGO	ADG
BOA*	2	$g_2^{\min}(s(x)), g_2^{\min}(s_{\text{goal}})$	$g_2^{\min}(s(x)), g_2^{\min}(s_{\text{goal}})$	✓	✓	✗
NAMOA*	≥ 2	$\mathbf{G}_{\text{cl}}(s(x)), \mathbf{G}_{\text{op}}(s(x)), \mathbf{G}_{\text{cl}}(s_{\text{goal}})$	$\mathbf{G}_{\text{cl}}(s_{\text{goal}})$	✗	✗	✗
NAMOA*dr	≥ 2	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x)), \mathbf{G}_{\text{op}}(s(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$	✓	✗	✗
EMOA*	≥ 2	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$	✓	✓	✗
LTMOA*	≥ 2	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$	$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$	✓	✓	✗
LazyLTMOA*	≥ 2		$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x)), \mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$	✓	✓	✓

Table 3: Summary of the dominance checks done by all algorithms and how they are performed. (DR: Dimensionality Reduction, AGO: Avoid $\mathbf{G}_{\text{op}}(s(x))$, ADG: Avoid Dominance checks in node Generation)

$\text{Tr}(\mathbf{g}(x_1)) \preceq \text{Tr}(\mathbf{g}(x_2))$ since $h_i(x_1) = h_i(s) = h_i(x_2)$ for all i. After x_1 is expanded and before x_2 is expanded, $\text{Tr}(\mathbf{g}(x_1))$ is added into $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$ (Line 14). Since $\text{Tr}(\mathbf{g}(x_1)) \preceq \text{Tr}(\mathbf{g}(x_2))$, which satisfies the pruning condition (Line 11), x_2 is not expanded, which contradicts the assumption. □

Corollary 1. *Two nodes x_1 and x_2 with the same state s that are sequentially expanded will satisfy $\mathbf{f}(x_1) \not\preceq \mathbf{f}(x_2)$.*

Proof Sketch: Since the truncated $\text{Tr}(\mathbf{f}(x_1))$ does not weakly dominate the truncated $\text{Tr}(\mathbf{f}(x_2))$ (Lemma 3), the non-truncated $\mathbf{f}(x_1)$ cannot weakly dominate $\mathbf{f}(x_2)$. □

Lemma 4. *Two nodes x_1 and x_2 with the same state s that are sequentially expanded will satisfy $\mathbf{f}(x_1) \not\succeq \mathbf{f}(x_2)$.*

Proof Sketch: Assume by contradiction that LTMOA* expands node x_1 with state s before node x_2 with state s, that it expands no node with state s after node x_1 and before node x_2, and that $\mathbf{f}(x_1) \succeq \mathbf{f}(x_2)$. We distinguish two cases:

- Node x_2 was in *Open* when node x_1 was extracted and expanded. Since LTMOA* extracts the node with the lexicographically smallest f-value, we distinguish two cases:
  - $f_i(x_1) < f_i(x_2)$ for some value i and $f_j(x_1) = f_j(x_2)$ for all $j < i$. In this case, $\mathbf{f}(x_1) \not\succeq \mathbf{f}(x_2)$, which contradicts the assumption.
  - $\mathbf{f}(x_1) = \mathbf{f}(x_2)$. After x_1 is expanded and before x_2 is expanded, $\text{Tr}(\mathbf{g}(x_1))$ is added into $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$ (Line 14). Further, since $\text{Tr}(\mathbf{g}(x_1))$ weakly dominates $\text{Tr}(\mathbf{g}(x_2))$, which satisfies the pruning condition (Line 11), node x_2 is not expanded, which contradicts the assumption.
- Node x_2 is not in *Open* when node x_1 was extracted and expanded. Therefore, there must exist a node x_3 in *Open* that is eventually expanded and it is an ancestor of node x_2. Since LTMOA* extracts the node with the lexicographically smallest f-value, we further distinguish two cases:
  - $f_i(x_1) < f_i(x_3)$ for some value i and $f_j(x_1) = f_j(x_3)$ for all $j < i$. In this case, $\mathbf{f}(x_1) \not\succeq \mathbf{f}(x_3)$. Combining this inequality and $\mathbf{f}(x_2) \succeq \mathbf{f}(x_3)$ (Lemma 1) yields $\mathbf{f}(x_1) \not\succeq \mathbf{f}(x_2)$, which contradicts the assumption.
  - $\mathbf{f}(x_1) = \mathbf{f}(x_3)$. Combining this equality with $\mathbf{f}(x_2) \succeq \mathbf{f}(x_3)$ (Lemma 1) yields $\mathbf{f}(x_1) \preceq \mathbf{f}(x_2)$. If $f_1(x_1) < f_1(x_2)$, then $\mathbf{f}(x_1) \not\succeq \mathbf{f}(x_2)$, which contradicts the assumption. Otherwise, if $f_1(x_1) = f_1(x_2)$, then $\text{Tr}(\mathbf{g}(x_1)) \preceq \text{Tr}(\mathbf{g}(x_2))$, which satisfies the pruning condition (Line 11) since $\text{Tr}(\mathbf{g}(x_1))$ is added into $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s)$ after node x_1 is expanded and before node x_3 is expanded (Line 14). Therefore, node x_2 is not expanded, which contradicts the assumption. □

Corollary 2. *Expanded nodes with the same state do not weakly dominate each other.*

Lemma 5. *If node x_1 with state s is weakly dominated by node x_2 with state s, then each node with the goal state in the subtree of the search tree rooted at node x_1 is weakly dominated by a node with the goal state in the subtree rooted at node x_2.*

Lemma 6. *When LTMOA* prunes a node x_1 with state s (on Lines 11 or 23) and this prevents it in the future from adding a node x_2 (with the goal state) to the solution set (on Line 16), then it can still add in the future a node (with the goal state) that weakly dominates node x_2 (on Line 16).*

Theorem 1. *LTMOA* computes a cost-unique Pareto-optimal solution set.*

Theorem 2. *LazyLTMOA* computes a cost-unique Pareto-optimal solution set.*

7 Experimental Results

The objective of our evaluation is twofold. First, we want to investigate the impact on performance when the dominance checking is partially and totally removed. Second, we want to compare the performance of state-of-the-art algorithms.

All algorithms were implemented from scratch in C using a standard binary heap for *Open*. The C++ implementation of EMOA* provided by Ren *et al.* [2022] is much slower than our C implementation (up to 20 times slower in average), so we include the results of the latter. We used a 2.80GHz Intel(R) Core(TM) i7-1165G7 CPU Linux laptop with 64GB of RAM. We use the New York map of the 9th DIMACS Implementation Challenge: Shortest Path[2] with the number of components in the cost function being 3, 4, and 5.[3] The original map has two components in the cost function for each edge that represent the travel distance (d) and the travel time (t). We used the economic cost (m), which combines toll and fuel consumption according to the road category of New York (NY) used in [Pulido *et al.*, 2015], as the third cost component for each edge. Additionally, we used a random integer number between 1 and 100 (r) and the number of edges (l) [Casas *et al.*, 2021] as the fourth and fifth components respectively. Following Hernández *et al.* [2023], the heuristic $\mathbf{h}$ corresponds to the exact cost for each individual component of reaching the goal state.

We first compare LTMOA*, LazyLTMOA*, along with two other variants of LazyLTMOA*, LazyLTMOA*-1 and LazyLTMOA*-2 which performs dominance checks against only

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

[3]We do not show results for two objectives as BOA* already performs dominance checks in $O(1)$.

	runtime(s)	$\lvert gen \rvert$	$\lvert exp \rvert$	$\lvert check \rvert$	$\lvert per \rvert$
	New York City (NY) with 4 Cost Components ($\lvert S \rvert = 264,346$, $\lvert E \rvert = 730,100$, avg $\lvert sols \rvert = 17,719$)				
LTMOA*	113.48	19,107,463	6,247,196	40,724,367,477	**174,722,665**
LazyLTMOA*-1	71.98	19,107,463	6,247,196	24,374,282,199	193,799,413
LazyLTMOA*-2	147.08	19,107,463	6,247,196	57,986,381,958	406,834,102
LazyLTMOA*	**71.24**	19,107,463	6,247,196	**20,173,843,419**	450,753,168

Table 4: Runtime (in secs), number of generated nodes ($\lvert gen \rvert$), number of expanded nodes ($\lvert exp \rvert$), number of domination checks ($\lvert check \rvert$) and the number of heap percolations ($\lvert per \rvert$) in average for the evaluated algorithms in 40 random instances.

	solved	t_{mean}	t_{max}	t_{min}	t_{median}
	NY with 4 Cost Components (avg $\lvert sols \rvert = 17,719$)				
NAMOA*dr	38/40	710.46(558.38)	3,600.00	0.15	3.11
EMOA*	38/40	560.19 (400.20)	3,600.00	0.15	3.04
LazyEMOA*	38/40	505.68 (342.83)	3,600.00	0.15	3.46
LazyLTMOA*-L	38/40	502.88 (339.87)	3,600.00	0.15	2.66
LazyLTMOA*-A	**40/40**	**71.24 (43.15)**	**807.37**	0.15	**1.28**
	NY with 5 Cost Components (avg $\lvert sols \rvert = 36,887$)				
LazyEMOA*	21/40	1,790.04 (152.45)	3,600.00	0.18	1,470.35
LazyLTMOA*-L	21/40	1,788.27 (149.08)	3,600.00	0.18	1,478.75
LazyLTMOA*-A	**28/40**	**1,358.21 (11.48)**	3,600.00	0.18	**98.69**
	NY with 3 Cost Components (avg $\lvert sols \rvert = 34,857$)				
LazyEMOA*	86/100	793.32 (263.03)	3,600.00	0.11	55.53
LazyLTMOA*-L	84/100	913.36 (401.62)	3,600.00	0.11	64.60
LazyLTMOA*-A	**93/100**	**517.92 (104.63)**	3,600.00	0.11	**28.36**

Table 5: Random instances (40 for 4 and 5 components, and 100 for 3 components) solved and statistics on runtimes t (in secs). When an algorithm times out after 3,600s, we use 3,600s in the calculations.

$\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$ and $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s(x))$, respectively, when a node x is generated. All algorithms use an array to implement the $\mathbf{G}_{\text{cl}}^{\text{Tr}}$ set. Table 4 shows the runtime, the number of generated nodes ($\lvert gen \rvert$), the number of expanded nodes ($\lvert exp \rvert$), the number of domination checks ($\lvert check \rvert$), and the number of heap percolations in $Open$ ($\lvert per \rvert$). We use 4 components (l-d-t-m) as the cost function. We can observe that all algorithms obtain the same number of nodes generated and nodes expanded. Also, since the number of domination checks can be in orders of magnitude greater than the number of heap percolations, we can observe that LazyLTMOA* performs the smallest number of domination checks and thus results in the best runtime (slightly better than LazyLTMOA*-1) even with the highest number of heap percolations. In contrast, LazyLTMOA*-2 has the worse runtime since it has the highest number of domination checks. In this version, all the generated nodes are checked on $\mathbf{G}_{\text{cl}}^{\text{Tr}}(s_{\text{goal}})$, that can be the biggest $\mathbf{G}_{\text{cl}}^{\text{Tr}}$ set.

Table 5 shows the number of instances solved within a time limit of 3,600s and the runtime statistics in NY map with 5 (l-d-t-m-r), 4 (l-d-t-m) and 3 (d-t-m) components. The number in the bracket of t_{mean} is the mean of runtime among the instances solved by all the algorithms. The evaluated algorithms are NAMOA*dr and EMOA* (only with 4 cost function), LazyEMOA*, LazyLTMOA* with the linked list implementation of $\mathbf{G}_{\text{cl}}^{\text{Tr}}$ (LazyLTMOA*-L) and LazyLTMOA* with the array implementation of $\mathbf{G}_{\text{cl}}^{\text{Tr}}$ (LazyLTMOA*-A). For 5 and 3 components, we only show the results of the fastest algorithms. We observe that LazyLTMOA*-A solves more instances and is faster than NAMOA*dr, EMOA*, LazyEMOA*, and LazyLTMOA*-L. Considering only the instances that all algorithms solved, LazyLTMOA*-A is up to $\times 13.5$, $\times 7.9$, and $\times 2.5$ faster than LazyEMOA* in NY map with 5, 4, and 3 components, respectively. We observe that the performance of LazyLTMOA*-L is similar to the performance of LazyEMOA* for 3 and 5 components. With 3 components LazyEMOA* is faster than LazyLTMOA*-L due mainly to logarithmic time complexity of LazyEMOA* for dominance check. However, LazyLTMOA*-A is faster than LazyEMOA* due to the following reasons. (1) using simple arrays instead of linked lists or AVL trees is more efficient for implementation. (2) LazyEMOA* uses an AVL tree to keep the $\mathbf{G}_{\text{cl}}^{\text{Tr}}$ set of each state. Each time when a new element is inserted in the $\mathbf{G}_{\text{cl}}^{\text{Tr}}$ set, the tree is updated by removing dominated vectors and inserting the new element, which may trigger several heavy AVL balancing procedures.

8 Conclusions

We presented LTMOA* and LazyLTMOA*, new multi-objective search algorithms that implement a more efficient dominance checking than EMOA* using simple array data structures and a lazier approach that remove dominance checking during node generation. The results show that LazyLTMOA* outperforms EMOA* up to an order of magnitude in runtime.

Acknowledgements

This research was supported by the United States-Israel Binational Science Foundation (BSF) grants number 2019703 and 2021643 and by the Israeli Ministry of Science & Technology grants number 3-16079 and 3-17385. The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant number 2121028. The research at the Chilean universities was supported by the National Center for Artificial Intelligence CENIA FB210017, Basal ANID, and the Centro Ciencia & Vida FB210008, Financiamiento Basal ANID.

References

[Ahmadi *et al.*, 2021] Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Bi-objective search with bi-directional A*. In *Proceedings of the European Symposium on Algorithms*, volume 204, pages 3:1–3:15, 2021.

[Bachmann *et al.*, 2018] Daniel Bachmann, Fritz Bökler, Jakob Kopec, Kira Popp, Björn Schwarze, and Frank Weichert. Multi-objective optimisation based planning of power-line grid expansions. *ISPRS International Journal of Geo-Information*, 7(7):258, 2018.

[Bronfman *et al.*, 2015] Andrés Bronfman, Vladimir Marianov, Germán Paredes-Belmar, and Armin Lüer-Villagra. The maximin hazmat routing problem. *European Journal of Operational Research*, 241(1):15–27, 2015.

[Casas *et al.*, 2021] Pedro Maristany de las Casas, Luitgard Kraus, Antonio Sedeño-Noda, and Ralf Borndörfer. Targeted multiobjective dijkstra algorithm. *arXiv preprint arXiv:2110.10978*, 2021.

[Fu *et al.*, 2019] Mengyu Fu, Alan Kuntz, Oren Salzman, and Ron Alterovitz. Toward asymptotically-optimal inspection planning via efficient near-optimal graph search. In *Proceedings of Robotics: Science and Systems*, 2019.

[Fu *et al.*, 2021] Mengyu Fu, Oren Salzman, and Ron Alterovitz. Computationally-efficient roadmap-based inspection planning via incremental lazy search. In *Proceedings of International Conference on Robotics and Automation*, pages 7449–7456, 2021.

[Hernández *et al.*, 2023] Carlos Hernández, William Yeoh, Jorge A. Baier, Han Zhang, Luis Suazo, Sven Koenig, and Oren Salzman. Simple and efficient bi-objective search algorithms via fast dominance checks. *Artificial Intelligence*, 314:103807, 2023.

[Li *et al.*, 2015] B. Li, J. Li, K. Tang, and X. Yao. Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys*, 48(1):1–35, 2015.

[Mandow and De La Cruz, 2010] Lawrence Mandow and José Luis Pérez De La Cruz. Multiobjective A* search with consistent heuristics. *Journal of the ACM*, 57(5):27:1–27:25, 2010.

[Mandow *et al.*, 2005] Lawrence Mandow, JL Pérez De la Cruz, et al. A new approach to multiobjective A* search. In *Proceedings of International Joint Conference of Artificial Intelligence*, pages 218–223, 2005.

[Pangilinan and Janssens, 2007] J. M. Pangilinan and G. Janssens. Evolutionary algorithms for the multiobjective shortest path problem. *World Academy of Science, Engineering and Technology*, 25:205–210, 2007.

[Pulido *et al.*, 2015] Francisco-Javier Pulido, Lawrence Mandow, and José-Luis Pérez-de-la-Cruz. Dimensionality reduction in multiobjective shortest path search. *Computers & Operations Research*, 64:60–70, 2015.

[Ren *et al.*, 2022] Zhongqiang Ren, Richard Zhan, Sivakumar Rathinam, Maxim Likhachev, and Howie Choset. Enhanced multi-objective a* using balanced binary search trees. In *Proceedings of the International Symposium on Combinatorial Search*, pages 162–170, 2022.

[Salzman and Goldin, 2021] Oren Salzman and Boris Goldin. 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*, pages 149–158, 2021.

[Skyler *et al.*, 2022] S. Skyler, D. Atzmon, A. Felner, O. Salzman, H. Zhang, S. Koenig, W. Yeoh, and C. Hernández. Bounded-cost bi-objective heuristic search. In *Proceedings of the International Symposium on Combinatorial Search*, pages 239–243, 2022.

[Zhang *et al.*, 2022a] H. Zhang, O. Salzman, T. K. S. Kumar, A. Felner, C. Hernández, and S. Koenig. Anytime approximate bi-objective search. In *Proceedings of the International Symposium on Combinatorial Search*, pages 199–207, 2022.

[Zhang *et al.*, 2022b] H. Zhang, O. Salzman, TK S. Kumar, A. Felner, C. Hernández, and S. Koenig. A*pex: Efficient approximate multi-objective search on graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 394–403, 2022.