Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

Parallelizing Multi-objective A* Search

Saman Ahmadi[1], **Nathan R. Sturtevant**[2], **Andrea Raith**[3], **Daniel Harabor**[4], **Mahdi Jalili**[1]
[1] School of Engineering, RMIT University, Australia
[2] Department of Computing Science, University of Alberta, Canada
[3] Department of Engineering Science, University of Auckland, New Zealand
[4] Department of Data Science and AI, Monash University, Australia
saman.ahmadi@rmit.edu.au, nathanst@ualberta.ca, a.raith@auckland.ac.nz
daniel.harabor@monash.edu, mahdi.jalili@rmit.edu.au

Abstract

The Multi-objective Shortest Path (MOSP) problem is a classic network optimization problem that aims to find all Pareto-optimal paths between two points in a graph with multiple edge costs. Recent studies on multi-objective search with A* (MOA*) have demonstrated superior performance in solving difficult MOSP instances. This paper presents a novel search framework that allows efficient parallelization of MOA* with different objective orders. The framework incorporates a unique upper-bounding strategy that helps the search reduce the problem's dimensionality to one in certain cases. Experimental results demonstrate that the proposed framework can enhance the performance of recent A*-based solutions, with the speed-up proportional to the problem dimension.

Introduction

The Multi-objective Shortest Path problem (MOSP), a challenging network optimization problem, aims to find all Pareto-optimal paths between a given pair of nodes in a network where each edge is associated with multiple attributes. MOSP can emerge in various real-world problems. Applications are multi-objective path planing for mobile robots (Ren, Rathinam, and Choset 2021), multi-objective route selection problem for unmanned air vehicles (Tezcaner and Köksalan 2011), and multi-objective routing for airport ground movement with factors such as taxi time, fuel consumption and emissions (Weiszer, Burke, and Chen 2020).

Salzman et al. (2023) presented an overview of recent advances in bi-objective and multi-objective search, highlighting the significant progress made by heuristic search in enhancing the efficiency of MOA* (Stewart and White III 1991). The EMOA* (Ren et al. 2022), TMDA (Maristany de las Casas et al. 2023) and LTMOA* (Hernández et al. 2023) algorithms are three state-of-the-art approaches that utilize best-first search to solve point-to-point MOSP more efficiently. The LTMOA* algorithm, in particular, is shown to perform up to an order of magnitude faster than EMOA* due to its more efficient dominance checking rules. Nevertheless, LTMOA*'s performance in large networks degrades when the number of objectives increases, leaving some difficult MOSP instances unsolved even after a one-hour runtime. Recently, the NWMOA* algorithm (Ahmadi et al. 2024a) has demonstrated faster performance than previous approaches, including the improved variant of the NAMOA*$_{dr}$ algorithm (Pulido, Mandow, and Pérez-de-la-Cruz 2015) studied in Maristany de las Casas et al. (2023).

In many domains, performance of planning algorithms may be impacted by slight modifications of the search setting, such as changes on tie-breaking rules or order of operators (Knight 1993; Howe and Dahlman 2002; Ahmadi et al. 2024b). In the case of multi-objective search, while MOA* offers a robust framework for optimally solving MOSP in large graphs, the order of objectives can have a "*dramatic effect on the algorithm's running times*" (Salzman et al. 2023). Although a good-performing ordering can sometimes be obtained empirically, as in Hernández et al. (2023), there is no guarantee that such ordering performs best in all instances. Incorporating multiple objective orderings in a parallel setting can be seen as a potential solution to the above shortcoming. Despite being well-studied for single-objective search in various settings (Valenzano et al. 2010; Zhou and Zeng 2015; Fukunaga et al. 2018), parallelization remains largely underexplored in the literature on multi-objective search. While existing approaches mainly focus on parallelization of search procedures (Sanders and Mandow 2013; Erb, Kobitzsch, and Sanders 2014; Ulloa et al. 2024), the bi-objective search algorithm BOBA* (Ahmadi et al. 2021) and its weight-constrained adaptation (Ahmadi et al. 2022) offer a parallel framework that executes two concurrent searches with different objective orderings. Although this parallelization strategy has been shown to be effective in reducing computation time, its applicability to MOA* remains uncertain.

This research proposes a parallel search framework for MOA*, representing, to the best of our knowledge, the first attempt to parallelize multi-objective search for more than two objectives. Inspired by the search scheme of BOBA*, our proposed approach runs multiple MOA* searches in parallel, but on different objective orderings. Central to our framework is an innovative upper-bounding technique that not only shrinks the scaled search space but also reduces the problem's dimensionality to one under specific conditions. Results demonstrate the effectiveness of our parallel framework in achieving scaled acceleration of the MOA* search, with speed-ups proportional to the number of objectives, achieving up to an order of magnitude improvement on challenging MOSP instances.

Notation and Problem Formulation

Consider a directed graph $G = (S, E)$ with a finite set of states S and a set of edges $E \subseteq S \times S$ with every edge $e \in E$ comprising $k \in \mathbb{N}$ attributes that can be accessed via the cost function $\mathbf{cost} : E \rightarrow \mathbb{R}^k$, that is, we have $\mathbf{cost} = (cost_1, cost_2, \ldots, cost_k)$ as a form of vector. A path π is a sequence of states $u_i \in S$ with $i \in \{1, \ldots, n\}$ and $(u_i, u_{i+1}) \in E | \; i < n$. The $\mathbf{cost}$ vector of the path is then the sum of corresponding attributes on all the edges constituting the path, namely $\mathbf{cost}(\pi) = \sum_{i=1}^{n-1} \mathbf{cost}(u_i, u_{i+1})$. The MOSP problem aims to find a set of Pareto-optimal paths between a given pair of $start \in S$ and $goal \in S$. A path π^* is Pareto efficient if there does not exist any other path π' that simultaneously satisfies $cost_i(\pi') < cost_i(\pi^*)$ and $cost_j(\pi') \leq cost_j(\pi^*)$ for any $i, j \in \{1, \ldots, k\} | i \neq j$.

In the context of A* search, we define our search objects as *nodes*. A node x is a tuple that contains the main information on the partial path from $start$ to state $s(x)$, where $s(x)$ is a function that returns the state associated with x. Node x contains: i) the cost vector $\mathbf{g}(x)$, representing the $\mathbf{cost}$ of a concrete path from $start$ to state $s(x)$; ii) the cost vector $\mathbf{f}(x)$, denoting the $\mathbf{cost}$ estimate of a complete path from $start$-$goal$ via the partial path represented by x; iii) the reference node $parent(x)$, indicating the parent node of x.

All operations of the cost vectors are considered to be performed element-wise. For example, we define $\mathbf{g}(x)+\mathbf{g}(y)$ as $(g_1(x) + g_1(y), \ldots, g_k(x) + g_k(y))$. We use $\preceq$ or $\prec$ symbols in direct comparisons of cost vectors, e.g. $\mathbf{f}(x) \preceq \mathbf{f}(y)$ denotes $f_i(x) \leq f_i(y)$ for all $i \in \{1, \ldots, k\}$. Further, the operator $\mathrm{Tr}(\mathbf{v})$ truncates the first cost of the vector $\mathbf{v}$, and Tr^λ denotes λ consecutive truncations, e.g., $\mathrm{Tr}(\mathbf{f}(x)) = (f_2(x), \ldots, f_k(x))$ and $\mathrm{Tr}^2(\mathbf{f}(x)) = (f_3(x), \ldots, f_k(x))$.

Definition Cost vector $\mathbf{v}$ is weakly dominated by cost vector $\mathbf{v}'$ if we have $\mathbf{v}' \preceq \mathbf{v}$; $\mathbf{v}$ is dominated by $\mathbf{v}'$ if $\mathbf{v}' \preceq \mathbf{v}$ and $\mathbf{v}' \neq \mathbf{v}$; $\mathbf{v}$ is not dominated by $\mathbf{v}'$ if $\mathbf{v}' \not\preceq \mathbf{v}$. Node y is weakly dominated by node x if $\mathbf{f}(x) \preceq \mathbf{f}(y)$.

Multi-objective Search with A*

Multi-objective pathfinding with A* involves a systematic search by *expanding* nodes in best-first order, that is, the search is guided by paths showing smallest $start$-$goal$ cost estimates. These estimates are traditionally established as $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}(s(x))$ where a consistent heuristic function $\mathbf{h} : S \rightarrow \mathbb{R}^k$ estimates lower bounds on the $\mathbf{cost}$ of extended paths to $goal$ (Hart, Nilsson, and Raphael 1968).

To elaborate on the key steps of the search, we have provided in Algorithm 1 a pseudocode of the recent MOA* methods to MOSP, namely LTMOA* and NWMOA*. The search starts with initializing a pair of node sets. The first set, $Open$, is responsible for maintaining unexplored nodes in best-first order. The second set, $Sols$, stores all discovered solution nodes during the search, and is returned as output. Next, it initializes, for every state of the graph, a list called G^{Tr}. For a typical state u, $\mathrm{G}^{\mathrm{Tr}}(u)$ always contains non-dominated truncated cost vectors expanded with u. To begin the search, the algorithm then initializes a node with the $start$ state and inserts it into $Open$.

Algorithm 1: Multi-objective Search with A*

Input: A MOSP problem (G, $\mathbf{cost}$, $\mathbf{h}$, $start$, $goal$)
Output: A cost-unique Pareto-optimal solution set

1 $Open \leftarrow \emptyset$, $Sols \leftarrow \emptyset$
2 $\mathrm{G}^{\mathrm{Tr}}(u) \leftarrow \emptyset \;\; \forall u \in S$
3 $x \leftarrow$ new node with $s(x) = start$
4 $\mathbf{g}(x) \leftarrow \mathbf{0}$, $\mathbf{f}(x) \leftarrow \mathbf{h}(start)$, $parent(x) \leftarrow null$
5 Add x to $Open$
6 **while** $Open \neq \emptyset$ **do**
7 Extract node x from $Open$ with the smallest $\mathbf{f}$-value
8 **if** `IsDominated`$(\mathrm{Tr}(\mathbf{g}(x)), \mathrm{G}^{\mathrm{Tr}}(s(x)))$ **or** `IsDominated`$(\mathrm{Tr}(\mathbf{f}(x)), \mathrm{G}^{\mathrm{Tr}}(goal))$ **then**
9 **continue**
10 `RemoveDominated`$(\mathrm{Tr}(\mathbf{g}(x)), \mathrm{G}^{\mathrm{Tr}}(s(x)))$
11 Add $\mathrm{Tr}(\mathbf{g}(x))$ to $\mathrm{G}^{\mathrm{Tr}}(s(x))$
12 **if** $s(x) = goal$ **then**
13 Add x to $Sols$
14 **continue**
15 **for each** $t \in Succ(s(x))$ **do**
16 $y \leftarrow$ new node with $s(y) = t$
17 $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{cost}(s(x), t)$
18 $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$
19 $parent(y) \leftarrow x$
20 **if** `IsDominated`$(\mathrm{Tr}(\mathbf{g}(y)), \mathrm{G}^{\mathrm{Tr}}(t))$ **or** `IsDominated`$(\mathrm{Tr}(\mathbf{f}(y)), \mathrm{G}^{\mathrm{Tr}}(goal))$ **then**
21 **continue**
22 Add y to $Open$
23 **return** $Sols$

Each iteration of the algorithm starts at line 6. Let x be a node offering the smallest $\mathbf{f}$-value among nodes in $Open$. Within each iteration, the algorithm attempts to extend x towards $goal$. However, to avoid processing unpromising nodes, x is checked for dominance through the `IsDominated` procedure (line 8). The dominance rules are: i) whether x is weakly dominated by any node previous expanded with $s(x)$; ii) whether x can be weakly dominated by any current solution. Weakly dominated nodes can be safely pruned, as their expansion would not lead to any cost-unique optimal solution. Since nodes in MOA* are explored in non-decreasing order of their primary cost estimate (here f_1), dominance checks can be done more efficiently by comparing the non-primary costs only. Algorithm 1 utilizes this dimension reduction technique within its dominance rules by truncating the first cost of the vector using the $\mathrm{Tr}(\cdot)$ operator. Given $\mathrm{G}^{\mathrm{Tr}}(s(x))$ containing the (non-dominated) truncated cost vectors of previous nodes explored with $s(x)$, node x is pruned if $\mathrm{Tr}(\mathbf{g}(x))$ is weakly dominated by a cost vector in $\mathrm{G}^{\mathrm{Tr}}(s(x))$. Similarly, x is pruned if its truncated $\mathbf{cost}$ estimate $\mathrm{Tr}(\mathbf{f}(x))$ is weakly dominated by that of any solution in $\mathrm{G}^{\mathrm{Tr}}(goal)$. Otherwise, if x is not pruned by either of the rules, its truncated $\mathbf{cost}$ vector will be stored for the purpose of future dominance tests with $s(x)$ (line 11), but before that, the algorithm removes from $\mathrm{G}^{\mathrm{Tr}}(s(x))$ all vectors dominated by $\mathrm{Tr}(\mathbf{g}(x))$. This operation ensures the vectors of $\mathrm{G}^{\mathrm{Tr}}(s(x))$ always remain non-dominated.

Once proven non-dominated, x will be either added to $Sols$ if $s(x)$ is the $goal$ state (line 12), or expanded. The expansion operation *generates* a set of successor states, each denoted $Succ(s(x))$. Let y be a descendant node of x. To reduce the queue load, MOA* can check y against dominance rules before inserting it into $Open$ for further expansions (line 20). However, the literature has documented faster MOA* performance with *lazy* dominance checks, where nodes are checked against dominance rules only after they are extracted from $Open$. Finally, MOA* terminates when there is no node in $Open$ to explore, returning $Sols$ as a set of cost-unique solution nodes to the given MOSP instance.

LTMOA* vs. NWMOA*: Both algorithms follow the outline of MOA* in Algorithm 1, but they differ in their node exploration methods in three aspects:
i) **Ordering of nodes:** The priority queue in LTMOA* orders nodes based on their $\mathbf{cost}$ lexicographically, whereas NWMOA* explores nodes in the order of primary cost only. The former reduces node expansions, with no expansions of dominated nodes. The latter, however, may incur extra expansions but does not need to handle tie-breaking.
ii) **Dominance check:** LTMOA* stores the truncated cost vector of previous expansions in no specific order, whereas NWMOA* stores them in lexicographical order. The former necessitates a linear scan over all vectors of the $\mathrm{G^{Tr}}$ list to ensure the new vector is non-dominated but allows non-dominated vectors to be added to the list more efficiently. The latter, however, allows for partial traversal over vectors of the $\mathrm{G^{Tr}}$ list expansions, but incurs sorting overhead.
iii) **Quick pruning:** NWMOA* utilizes an additional dominance test that enables quick pruning of some nodes. At every state it caches the most recent non-dominated truncated cost vector, which is used as a first (more-informed) candidate for quick dominance pruning before attempting the full dominance tests. This technique has proven to be effective in reducing the total number of dominance checks drastically, enhancing the search performance.

A Parallel MOA* Framework

This section describes our novel parallelized MOA* solution to MOSP. The idea is straightforward: let MOA* be guided with more than one objective order at a time. Here, we are interested in a set of orderings that allows every objective to appear as the primary cost in one search. Algorithm 2 presents the higher level of the proposed parallel MOA*. Given a k-dimensional MOSP instance, the algorithm uses k CPU threads to run k individual MOA* searches in parallel such that the i-th thread , $i \in \{1, \ldots, k\}$, is provided with a cost and heuristic function that return $cost_i$ and h_i as primary cost and heuristic, respectively (lines 3-5). One such set of orderings can simply be all cyclic permutations of the costs. For example, in a three-dimensional instance ($k = 3$), the second thread can use a cost function that returns edge $\mathbf{cost}$ in $(cost_2, cost_3, cost_1)$ order with $cost_2$ as the primary cost. For this thread, we provide MOA*'s heuristic function $\mathbf{h}$ in (h_2, h_3, h_1) order. Since each MOA* search is complete, the parallel loop can terminate as soon as one thread finishes its search.

Algorithm 2: Parallel MOA*- Higher Level

Input: A MOSP Problem (G, $start$, $goal$, k)
Output: A cost-unique Pareto-optimal solution set
1 $\mathbf{Sols} \leftarrow \emptyset^k$, $\bar{\mathbf{f}} \leftarrow \infty^k$
2 **for** $i \in \{1, \ldots, k\}$ **do in parallel**
3 $\quad \mathbf{c} \leftarrow$ cost function with $cost_i$ as primary cost
4 $\quad \mathbf{h} \leftarrow$ heuristic function corresponding to $\mathbf{c}$
5 $\quad$ Parallelized MOA* on (G, $\mathbf{c}$, $\mathbf{h}$, $start$, $goal$, $Sols_i$)
6 **return** Unique($\bigcup_{i=1}^{k} Sols_i$)

Necessity of upper bounding: Although the simple parallelization above allows for more than one objective ordering to be involved, it does not necessarily reduce the overall computation time if the searches are conducted independently, primarily due to algorithmic overhead driven by multiple computationally demanding MOA* searches. More precisely, upon the termination of the search in one thread, we have captured all optimal solutions, rendering the search effort of other threads superfluous. To reduce this overhead, the search in each thread could be informed with the optimal solutions discovered in the other threads, so they can prune unpromising paths not leading to a $start$-$goal$ path better than *any* discovered optimal solution as global upper bounds. Nonetheless, this method is not efficient in practice, essentially because upper bounding via all discovered solutions (in linear-time fashion) becomes costly in the absence of a unified objective ordering. Consequently, effective upper bounding remains a crucial bottleneck for efficient parallelization of MOA*.

Our approach: To address the above-mentioned shortcoming, this research designs a novel upper-bounding mechanism that effectively reduces the search effort in each parallelized MOA* by establishing a unique information-sharing pipeline between the threads. Let $(f_1, f_2, \ldots, f_k)$ be the $\mathbf{cost}$ of the solution node x obtained in the first thread guided by $cost_1$. The search in this thread prunes paths with estimated $(cost_2, \ldots, cost_k)$ no better than $(f_2, \ldots, f_k)$ due to the first dimension already being non-decreasing. Now, assume that the second thread, guided by $cost_2$, has just extracted a node y with the primary cost of f'_2. We claim that the dimension in the dominance pruning with x (in the first thread) can be further reduced if we observe $f_2 \leq f'_2$, that is, as soon as we see f_2 within the explored range of the second thread. In this case, we just need to check the estimated $(cost_3, \ldots, cost_k)$ of paths in the first thread against $(f_3, \ldots, f_k)$ with the second dimension also removed. This pruning is correct, as it basically means extension of paths with estimated costs no smaller than $(f_3, \ldots, f_k)$ – in any dimension – would definitely lead to a $start$-$goal$ path no better than either x or one of the optimal solutions obtained in the second thread. Note that when MOA* in the second thread extracts y, it guarantees that all solutions with $cost_2$ smaller than f'_2 are already captured.
Constant-time upper bounding: The truncation strategy above can be applied to the other threads to potentially reduce the dimension in this upper-bound pruning to *one*, enabling a fast $O(1)$ time pruning rule against a subset of so-

lutions. For example, if we observe the f_3-value of the doubly truncated cost vector $(f_3, \ldots, f_k)$ within the explored range of the third thread (guided by $cost_3$), with a similar reasoning, we can perform one more truncation and use the vector $(f_4, \ldots, f_k)$ as a tighter upper bound instead. It then becomes clear that $k-1$ consecutive truncations of the cost vector $(f_1, \ldots, f_k)$ yields f_k as a scalar upper bound on the k-th objective of the problem. This implies that the dimensionality of the problem in upper-bound pruning can potentially be reduced to one, as nodes only need to be checked against a scalar upper bound if a solution node's cost vector has already undergone $k-1$ consecutive truncations.

Global upper bound and early termination: Cost vectors truncated $k-1$ times can serve as a *global* upper bound, and also a termination criterion in parallelized MOA*. Here, 'global' indicates that each scalar upper bound is not restricted to the thread in which it was obtained but can be utilized across the entire search. Let $\overline{\mathbf{f}} = (\overline{f_1}, \ldots, \overline{f_k})$ represent a vector of scalar upper bounds. In case of objective ordering with cyclic permutations of costs, $\overline{\mathbf{f}}$ can be fully leveraged, as each thread is able to capture one of the scalar upper bounds. Thus, we can prune nodes (in any of the parallel searches) not respecting at least one of the established global upper bounds. Initially set to infinity, these global upper bounds can not only help other threads to strengthen their upper bound pruning, but enable them to terminate early as soon as their primary cost exceeds the corresponding global upper bound. For instance, the global upper bound $\overline{f_k}$ can be used in the k-th thread to terminate the search early as soon as a node with a primary cost no smaller than $\overline{f_k}$ is extracted, given that nodes in the k-th thread are always explored in monotonically non-decreasing order of their f_k-value.

To implement our upper bounding strategy, we initialize a list of solution sets $\mathbf{Sols}$ as $\{Sols_1, \ldots, Sols_k\}$, and a global upper bound vector $\overline{\mathbf{f}}$, both shared between all threads (line 1 of Algorithm 2). We now explain the necessary changes in MOA* to incorporate the upper bounding procedure. Algorithm 3 shows the detailed steps involved in our i-th parallelized MOA* led by $cost_i$. There are three key changes:

i) To facilitate multidimensional truncations, $\mathbf{G^{Tr}}$ now organizes truncated cost vectors by storing those with the same cardinality together in separate lists. Thus, we have $\mathbf{G^{Tr}}$ as $\{\mathrm{G}_1^{\mathrm{Tr}}, \ldots, \mathrm{G}_{k-1}^{\mathrm{Tr}}\}$ with the index of G^{Tr} denoting the number of truncations. For example, $\mathrm{G}_2^{\mathrm{Tr}}$ contains (non-dominated) doubly truncated costs vectors. Although the multidimensional truncation is applied to the cost vectors of the solution nodes (associated with $goal$) only, we generalize it to all states for the sake of simplicity in our algorithm description (line 2).

ii) Dominance check against previous solutions (our upper bounds) is now done through the $\texttt{IsDominated}_{\texttt{MD}}$ procedure (lines 8, 21). As presented in Algorithm 4, the cost vector $\mathbf{v}$ is first checked against the global upper bound $\overline{\mathbf{f}}$, where $f_i(\mathbf{v})$ denotes the f_i-value of the cost vector. There will then be $k-1$ phases of dominance checks if $\mathbf{v}$ is deemed within the global upper bound. In each phase, the λ truncated cost vector is checked against a corresponding set of truncated costs in $\mathbf{G^{Tr}}(goal)$, namely $\mathrm{G}_\lambda^{\mathrm{Tr}}(goal)$.

Algorithm 3: Parallelized MOA* Search

Input: Problem (G, $\mathbf{cost}$, $\mathbf{h}$, $start$, $goal$) and a set $Sols$
Output: $Sols$ updated with a subset of optimal solutions

1 $Open \leftarrow \emptyset$
2 $\mathbf{G^{Tr}}(u) \leftarrow \emptyset^{k-1} \ \ \forall u \in S$
3 $x \leftarrow$ new node with $s(x) = start$
4 $\mathbf{g}(x) \leftarrow \mathbf{0}$, $\mathbf{f}(x) \leftarrow \mathbf{h}(start)$, $parent(x) \leftarrow null$
5 Add x to $Open$
6 **while** $Open \neq \emptyset$ **do**
7 Extract node x from $Open$ with the smallest $\mathbf{f}$-value
8 **if** $\texttt{IsDominated}(\mathrm{Tr}(\mathbf{g}(x)), \mathrm{G}_1^{\mathrm{Tr}}(s(x)))$ **or** $\texttt{IsDominated}_{\texttt{MD}}(\mathbf{f}(x), \mathbf{G^{Tr}}(goal))$ **then**
9 **continue**
10 $\texttt{RemoveDominated}(\mathrm{Tr}(\mathbf{g}(x)), \mathrm{G}_1^{\mathrm{Tr}}(s(x)))$
11 Add $\mathrm{Tr}(\mathbf{g}(x))$ to $\mathrm{G}_1^{\mathrm{Tr}}(s(x))$
12 **if** $s(x) = goal$ **then**
13 $\texttt{UpdateUpperBound}(\mathbf{G^{Tr}}(goal))$
14 Add x to $Sols$
15 **continue**
16 **for each** $t \in Succ(s(x))$ **do**
17 $y \leftarrow$ new node with $s(y) = t$
18 $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{cost}(s(x), t)$
19 $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$
20 $parent(y) \leftarrow x$
21 **if** $\texttt{IsDominated}(\mathrm{Tr}(\mathbf{g}(y)), \mathrm{G}_1^{\mathrm{Tr}}(t))$ **or** $\texttt{IsDominated}_{\texttt{MD}}(\mathbf{f}(y), \mathbf{G^{Tr}}(goal))$ **then**
22 **continue**
23 Add y to $Open$
24 **return**

Algorithm 4: $\texttt{IsDominated}_{\texttt{MD}}$

Input: A vector $\mathbf{v}$ and a list of sets $\mathbf{V}$
Output: $true$ if $\mathbf{v}$ is weakly dominated, $false$ otherwise

1 **for** $i \in \{1, \ldots, k\}$ **do**
2 **if** $f_i(\mathbf{v}) \geq \overline{f_i}$ **then return** $true$
3 **for** $\lambda \in \{1, \ldots, k-1\}$ **do**
4 **if** $\texttt{IsDominated}(\mathrm{Tr}^\lambda(\mathbf{v}), \mathrm{V}_\lambda)$ **then return** $true$
5 **return** $false$

iii) Upon extraction of a non-dominated solution node, the search attempts to improve the upper bounds stored in $\mathbf{G^{Tr}}(goal)$ through the $\texttt{UpdateUpperBound}$ procedure (line 13), depending on the search progress in the other threads. Central to our upper bounding strategy, this procedure is detailed in Algorithm 5. In each thread, starting from the very first $\mathrm{G}_1^{\mathrm{Tr}}(goal)$ set, the procedure checks vectors of each set indexed by $\lambda \in \{1, \ldots, k-2\}$ for a further truncation. Let i be the primary (first appearing) cost index of the vector $\mathbf{v}$ in $\mathrm{G}_\lambda^{\mathrm{Tr}}(goal)$. The procedure then retrieves the most recent solution of the i-th thread from $Sols_i$ (or alternatively, the most recently extracted node in the i-th thread).

The vector $\mathbf{v}$ can be further truncated if we observe that its f_i-value is no larger than the f_i-value of the retrieved solution (or the recently extracted) node from the i-th thread. The truncation involves: removing the vector from its current set

Algorithm 5: `UpdateUpperBound`

Input: List of sets $\mathbf{V}$
Output: List of sets $\mathbf{V}$ updated

```
1  for λ ∈ {1, ..., k − 2} do
2      i ← index of the primary cost of vectors in V_λ
3      if Sols_i = ∅ then continue
4      z ← The most recent solution node in Sols_i
5      for v ∈ V_λ do
6          if f_i(v) ≤ f_i(z) then
7              Remove v from V_λ
8              if IsDominated(Tr(v), V_{λ+1}) then
9                  continue
10             RemoveDominated(Tr(v), V_{λ+1})
11             Add Tr(v) to V_{λ+1}
12 if V_{k−1} ≠ ∅ then
13     i ← index of the scalar cost in V_{k−1}
14     f̄_i ← the scalar cost in V_{k−1}
15 return
```

$\mathrm{G}^{\mathrm{Tr}}_{\lambda}$, truncating it, and inserting it into the next set $\mathrm{G}^{\mathrm{Tr}}_{\lambda+1}$, while ensuring the vectors of the set remain non-dominated after the new vector is added. Thus, the last set $\mathrm{G}^{\mathrm{Tr}}_{\mathrm{k}-1}$ is either empty or contains a single scalar. To keep the vectors of each set non-dominated, we first check the newly truncated vector against the existing candidates in $\mathrm{G}^{\mathrm{Tr}}_{\lambda+1}$. If the new vector is dominated, it does not need to be added to the set (line 8 of Algorithm 5). Otherwise, the newly truncated vector is non-dominated and should be added to the set. To keep our upper-bounding mechanism efficient, we remove from $\mathrm{G}^{\mathrm{Tr}}_{\lambda+1}$ all its vectors weakly dominated by the new vector (line 10). Compared with the single truncation technique utilized in the recent MOA* approaches, removal of dominated vectors from each set allows our upper-bounding method to reduce the total number of costs vectors stored in $\mathbf{G}^{\mathbf{Tr}}(goal)$. The newly truncated vector will be added into $\mathrm{G}^{\mathrm{Tr}}_{\lambda+1}$ if it is deemed non-dominated (line 11). In the end, if $\mathrm{G}^{\mathrm{Tr}}_{\mathrm{k}-1}$ is not empty, the procedure utilizes the (single) scalar upper bound obtained through $k-1$ truncations to update the corresponding global upper bound (lines 12-14).

The following example further elaborates on our proposed multidimensional truncation technique and demonstrates its effectiveness in reducing the search space.

Example: Consider a MOSP instance with four objectives ($k = 4$) to be solved with our parallel framework. There will be four threads, each guided by one of the objectives as a primary cost. The first thread takes the conventional $(cost_1, \ldots, cost_4)$ order. For the search in this thread, assume we have already found six solution paths, all provided in Table 1(a). $cost_2$ and $cost_3$ are the primary cost of the second and third search, respectively. Assume we have $f_2 = 6$ for the most recently solution in the second thread, and also $f_3 = 7$ for the latest solution in the third thread. We now show how the first thread can use the search progress in the other threads to improve its upper bounds in $\mathbf{G}^{\mathbf{Tr}}(goal)$ via three consecutive truncation phases.

First truncation: MOA* conventionally stores the non-dominated truncated cost vector of the solutions in a separate list, shown in the left column of Table 1(b). We observe that, upon capturing the last solution (shown in red), the first thread removes the very first truncated vector (crossed-out) from the list as we have $(5, 6, 8) \preceq (7, 8, 8)$. Our method now extends the truncation process in additional phases.

Second truncation: There are three (singly) truncated vectors in the first column of Table 1(b) with their f_2-value no larger than the f_2-value of the most recent solution in the second thread, i.e., $f_2 \leq 6$. These vectors are truncated and consequently moved into a separate list containing doubly truncated vectors containing (f_3, f_4), as shown in Table 1(c). Once transferred, we realize that the second vector is dominated by the third one, as we have $(6, 8) \preceq (7, 9)$. Thus, the dominated vector is removed to keep the list non-dominated.

Third truncation: The middle column of Table 1(c) contains two doubly truncated vectors, one of which has an f_3-value no larger than the f_3-value of the retrieved solution of the third thread, i.e., $f_3 \leq 7$. This vector will be truncated and consequently moved to the list of triply truncated vectors, as shown in Table 1(d), yielding an upper bound of 8 on $cost_4$.

Global upper bound: Since the `UpdateUpperBound` procedure tracks scalar upper bounds obtained in each parallel search, the single scalar cost from the final truncation phase in Table 1(d) can now serve as a global upper bound on $cost_4$. Thus, we can have $\overline{f_4} \leftarrow 8$, meaning that nodes with f_4-value no smaller than $\overline{f_4}$ can be pruned in all threads. This upper bound can also serve as a termination criterion for the thread guided by $cost_4$, allowing the thread to halt once it extracts a node with an f_4-value of 8 or greater. Note that $\overline{f_4}$ may later be updated (reduced) as additional solution costs are truncated throughout the search.

Upper-bound pruning: With the multidimensional list $\mathbf{G}^{\mathbf{Tr}}(goal)$ updated, now assume the first thread has extracted a new node with $\mathbf{f} = (9, 4, 9, 5)$ in the next iteration. We can observe that this node would not be pruned by any truncated cost vector in Table 1(a) if we had only applied MOA*'s single truncation phase, potentially leading to finding *duplicate* solutions. Instead, let us check this new cost vector against the upper bounds obtained in Table 1(d) through consecutive truncations. The first truncation yields $(4, 9, 5)$. None of the two candidates in $\mathrm{G}^{\mathrm{Tr}}_{1}(goal)$, the left most column of Table 1(d), dominates the (singly) truncated cost vector. Thus, we further truncate the vector and obtain $(9, 5)$. Checking this doubly truncated vector against the only vector of $\mathrm{G}^{\mathrm{Tr}}_{2}(goal)$, the middle column of the table, we find it dominated due to having $(9, 4) \preceq (9, 5)$. Thus, the new node can be pruned via our upper bounding approach, reducing the expansion effort and search space.

Merging optimal solutions: Although each parallelized search in our framework is equipped with an upper-bounding technique that helps shrink the overlap between the search spaces, it is still possible for the searches to capture some duplicate solutions, and even dominated solutions (as in NWMOA*). This is because our pruning strategy does not rigorously check nodes against all solutions of other threads. Thus, upon the parallel loop termination (i.e., once one of the threads terminates), the last step in our framework

$Sols_1$
(f_1, f_2, f_3, f_4)
$(4,7,8,8)$
$(5,5,9,4)$
$(6,9,8,7)$
$(7,8,5,8)$
$(8,4,7,9)$
$(9,5,6,8)$

(a)

G_1^{Tr}	G_2^{Tr}	G_3^{Tr}
(f_2, f_3, f_4)	(f_3, f_4)	f_4
~~$(7,8,8)$~~		
$(5,9,4)$		
$(9,8,7)$		
$(8,5,8)$		
$(4,7,9)$		
$(5,6,8)$		

(b) First truncation phase

G_1^{Tr}	G_2^{Tr}	G_3^{Tr}
(f_2, f_3, f_4)	(f_3, f_4)	f_4
~~$(5,9,4)$~~	$(9,4)$	
$(9,8,7)$		
$(8,5,8)$		
~~$(4,7,9)$~~	~~$(7,9)$~~	
~~$(5,6,8)$~~	$(6,8)$	

(c) Second truncation phase

G_1^{Tr}	G_2^{Tr}	G_3^{Tr}
(f_2, f_3, f_4)	(f_3, f_4)	f_4
	$(9,4)$	
$(9,8,7)$		
$(8,5,8)$		
	~~$(6,8)$~~	8

(d) Third truncation phase

Table 1: (a) A sample scenario with six solutions in the thread guided by f_1 (the most recent solution shown in red). Assume $f_2 = 6$ and $f_3 = 7$ for the most recent solution of the second and third threads, respectively; (b) The cost vectors in $\mathbf{G^{Tr}}$ after the initial truncation in MOA*. The truncated vector of the recent solution dominates that of the first solution (crossed-out); (c) The second phase further truncates three of the vectors (crossed-out) due to having their $f_2$within the explored range of the second thread; (d) The last phase similarly truncates one of the (doubly truncated) vectors, yielding an upper bound of 8 on f_4.

involves merging the solutions and then returning the cost-unique ones (line 6 of Algorithm 2). One straightforward strategy for this task is to first sort all discovered solutions lexicographically by their **cost**, and then perform a linear traversal to remove any weakly dominated solutions.

Theoretical Results

This section formalizes the correctness of our framework.

Lemma 1 Let $\mathbf{v}$ be the cost vector of a solution node x truncated λ times, $\lambda \in \{0, \ldots, k-2\}$, in a parallelized MOA*, that is, $\mathbf{v} = \mathrm{Tr}^{\lambda}(\mathbf{f}(x))$. Also let i be the index of the primary (first appearing) cost of $\mathbf{v}$, and f' be the f_i-value of the most recent node extracted in a thread guided by $cost_i$. The vector $\mathbf{v}$ can be further truncated if we observe $f_i(x) \leq f'$, which yields pruning nodes whose $\lambda + 1$ truncated $\mathbf{f}$-vector is weakly dominated by $\mathrm{Tr}^{\lambda+1}(\mathbf{f}(x))$.

Proof Sketch We use mathematical induction.

Base case $\lambda = 0$: The very first phase of truncation here is in fact the standard (single) truncation in MOA*. $\lambda = 0$ means that the vector $\mathbf{v}$ has not been truncated yet, and the solution node x already belongs to the thread led by $cost_i$. Given that the f_i-value of the extracted nodes in the thread led by $cost_i$ are monotonically non-decreasing, we always have $f_i(x) \leq f'$. It follows that checking new nodes against the primary cost of $\mathbf{v}$ (for dominance and upper bounding) is not necessary, and thus $\mathbf{v}$ can be truncated.

Inductive case: Assume the pruning rule is correct for λ consecutive truncations. We now check the correctness of the pruning rule for the next truncation phase $\lambda + 1$. Let $\mathbf{v}'$ be the truncated cost vector of x after $\lambda+1$ truncations, i.e., we have $\mathbf{v}' = \mathrm{Tr}^{\lambda+1}(\mathbf{f}(x))$ and $f_i(x) \leq f'$, where i is the index of the recently truncated cost, or equivalently the index of the primary cost of $\mathbf{v}$. Assume, for contradiction, that y could lead to a cost-unique optimal solution but is wrongly pruned due to the recent $\lambda + 1$ truncation. y is pruned because we have observed $\mathrm{Tr}^{\lambda+1}(\mathbf{f}(x)) \preceq \mathrm{Tr}^{\lambda+1}(\mathbf{f}(y))$. We can distinguish two cases:

i) $f_i(x) \leq f_i(y)$: this immediately yields $\mathrm{Tr}^{\lambda}(\mathbf{f}(x)) \preceq \mathrm{Tr}^{\lambda}(\mathbf{f}(y))$, which means that the weak dominance rule holds and thus y has correctly been pruned based on the assumed pruning rule of the previous truncation phase λ.

ii) $f_i(y) < f_i(x)$: incorporating the truncation condition, i.e., $f_i(x) \leq f'$, we will have $f_i(y) < f'$. Given the correctness of MOA* and that the thread guided by $cost_i$ always explores nodes in monotonically non-decreasing order of their f_i-value, if y leads to a cost-unique optimal solution, it must have already been captured by the search. This is because its f_i-value would fall within the explored range of the thread led by $cost_i$, i.e., f'.

We observe that node y cannot lead to a cost-unique optimal solution in either of the cases above, which contradicts our initial assumption. Thus, we conclude that $\mathbf{v}$ can be safely truncated if $f_i(x) \leq f'$. □

Lemma 2 Let $\mathbf{v}$ be the cost vector of a solution node truncated λ times, $\lambda \in \{0, \ldots, k-1\}$. $\mathbf{v}$ is a global upper bound across all threads.

Proof Sketch Under the premises of Lemma 1, $\mathbf{v}$ can be used within the same thread to prune out-of-bounds paths. However, due to the overlap between search spaces in the parallel setting, it is likely that such paths will also be generated by other threads. Let y be a node, generated in either of the threads, whose vector of partial costs $\mathbf{v}'$ corresponds with the costs in $\mathbf{v}$. If $\mathbf{v} \preceq \mathbf{v}'$, with a similar reasoning, we can prune y while guaranteeing that there is at least one solution (in one of the threads) that weakly dominates y. □

While the cost vectors of discovered solutions in each thread conventionally serve as upper bounds for their corresponding MOA*, Lemma 2 demonstrates that, in our parallel setting, each individual truncated cost vector can potentially act as a global upper bound, which can be leveraged by any thread to further reduce the search space. In this study, we only use cost vectors truncated $k-1$ times and form a global upper-bound vector $\overline{\mathbf{f}} = (\overline{f_1}, \ldots, \overline{f_k})$. It follows that MOA* in the thread led by $cost_i$, $i \in \{1, \ldots, k\}$ can terminate early if it extracts a node with an f_i-value no smaller than $\overline{f_i}$, knowing that all unexplored nodes in this thread are guaranteed to violate the global upper bound.

Theorem 1 Parallelized MOA* returns a set of cost-unique optimal solution paths to a MOSP instance.

Method	$\|\mathcal{S}\|$	Runtime(s) Min.	Mean	Max.	Sp. Up	Mem. (GB)
NY with 3 cost components (avg($\|Sols\|$) = 5,090)						
NWMOA*	100	0.01	2.43	14.0	-	0.05
NWMOA*$_{par}$	100	0.01	1.70	9.2	1.31	0.11
LTMOA*	100	0.03	10.60	70.0	-	0.07
LTMOA*$_{par}$	100	0.02	7.34	47.7	1.39	0.12
NY with 4 cost components (avg($\|Sols\|$) = 86,134)						
NWMOA*	98	0.09	508.00	3600.0	-	0.50
NWMOA*$_{par}$	100	0.07	189.06	1558.5	3.01	1.52
LTMOA*	90	0.22	988.87	3600.0	-	0.63
LTMOA*$_{par}$	100	0.13	403.74	3282.9	2.91	1.68
NY with 5 cost components (avg($\|Sols\|$) = 158,898)						
NWMOA*	74	0.13	1405.49	3600.0	-	0.52
NWMOA*$_{par}$	91	0.08	745.43	3600.0	4.23	2.60
LTMOA*	68	0.34	1720.05	3600.0	-	0.66
LTMOA*$_{par}$	86	0.13	1034.35	3600.0	3.75	2.89

Table 2: Performance of algorithms over 100 instances in three scenarios. Runtime statistics are in seconds and $|\mathcal{S}|$ is the number of solved cases. We also report, for mutually solved cases, the average memory usage (in GB) and speed up achieved by parallelization w.r.t. the standard variant.

Proof Sketch Our framework uses k CPU threads to parallelize k individual MOA* on different objective orders, where each thread can solve MOSP optimally. We proved in Lemmas 1-2 why our proposed upper-bound pruning strategies are correct. Thus, we just need to show that Algorithm 2 returns a cost-unique optimal solution set when it terminates. The parallel loop stops as soon as one of the threads terminates. This termination criterion is correct, as it essentially means the thread has completed a full exploration of the search space. Each thread returns a subset of cost-unique optimal solutions. However, due to the overlap between the searches, and since our framework does not fully share upper bounds between the threads, weakly dominated solutions are likely to appear among the returned solution sets. Thus, the framework performs a separate yet straightforward sorting step to merge and remove such solutions (line 6 of Algorithm 2), returning only cost-unique optimal solutions. □

Experimental Results

This section evaluates the performance of our proposed parallel framework based on two recent MOA* algorithms: NWMOA*(Ahmadi et al. 2024a) and LTMOA* with lazy dominance tests (Hernández et al. 2023). We implemented all algorithms in C++ using the same data structure for maintaining search nodes, enabling a head-to-head comparison.

Benchmark setup: For the benchmark map, following the literature, we used the New York map from the 9th DIMACS Implementation Challenge: Shortest Paths[1] in scenarios with three to five cost components. The first and second costs of each edge are *distance* and *time*, respectively. The third cost is the positive height difference of the endpoints of each link, with height information obtained from Shuttle Radar Topography Mission[2]. For the fourth cost, as in Ren et al. (2022), we calculate the average (out)degree of the link, that is, the number of adjacent vertices of each end point. Motivated by hazardous material transportation (Erkut, Tjandra, and Verter 2007), this cost is designed as an indicator of how interconnected an edge is within the network. The more connections an edge has with other links, the higher the potential risk associated with transporting through the link. Following Maristany de las Casas et al. (2023), we assign a value of one to the fifth cost of each edge, with $cost_5$ of paths representing the total number of links (or intersections) traversed. We then evaluated all algorithms on 100 random $(start, goal)$ pairs from Ahmadi et al. (2024a) for three scenarios: $k = 3, 4, 5$.

Implementation: All C++ code was compiled using the GCC7.5 compiler with O3 optimization settings, and all experiments were conducted on an Intel Xeon Platinum 8488C processor with four CPU cores (eight threads with hyperthreading enabled) running at 2.4 GHz and with 16 GB of RAM under Linux environment with a one-hour timeout. For the parallel algorithms, we used appropriate synchronization mechanisms whenever updating shared parameters. In addition, we performed three runs of each algorithm and stored the run showing the median runtime. For a problem instance with k objectives, we set the parallel algorithms to work on k different orderings obtained by cyclic permutations of objectives. We then allocated one thread to each objective ordering to fully leverage parallel computation of solutions. Although the number of threads involved in the computation here is limited to the number of objectives, one can still incorporate available threads for running additional MOA* on a different set of objective orderings. While this extension allows for higher utilization of resources, the effectiveness of the framework in reducing the overall computation time should be investigated due to added algorithmic overhead. We will study this extension of our parallel framework in future work. Our code is publicly available[3].

Performance impact of parallelization: Since all algorithms use the same approach to compute heuristic functions, we focus on the performance of the main search only. Table 2 compares the performance of our parallelized algorithms against their standard version, which use lexicographical ordering of objectives. We report the number of solved cases ($|\mathcal{S}|$ out of 100) and runtime statistics (in seconds). The runtime of unsolved cases is considered to be the timeout. We also report, for mutually solved cases, the average speed up (Sp. Up) obtained over the standard variant, and average memory usage (in GB). We observe that, compared to the standard methods, the parallel variants:
i) consistently solve more instances. In the $k = 5$ scenario, LTMOA*$_{par}$ and NWMOA*$_{par}$ solve 18 and 17 more instances within the one-hour timeout, respectively;
ii) offer statistically considerable runtime improvement. In the $k = 4$ scenario, maximum runtime of NWMOA* is reduced from (above) 60 minutes to 26 minutes, and the average runtime from 8 to 3 minutes, approximately;

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

[2] https://www2.jpl.nasa.gov/srtm/

[3] https://bitbucket.org/s-ahmadi/multiobj

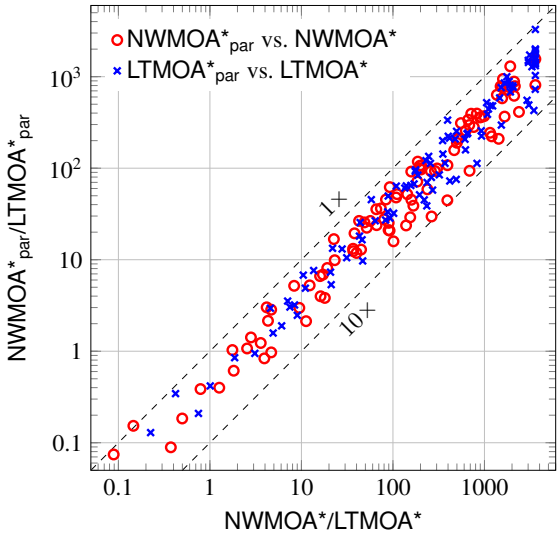

Figure 1: Runtime distribution of NWMOA* and LTMOA* versus their parallelized variant over instances with $k = 4$.

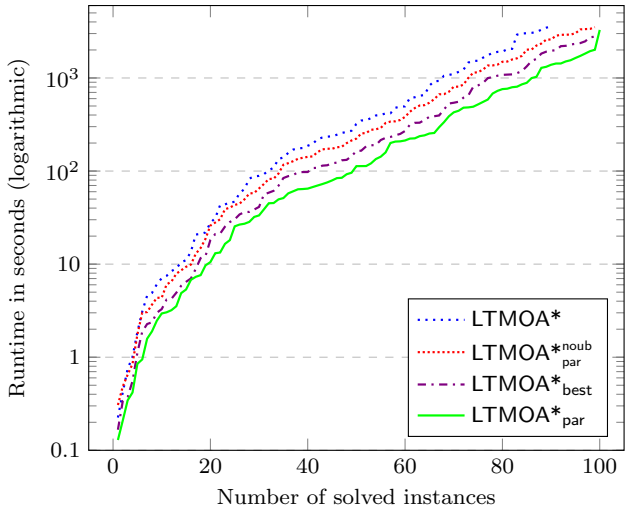

Figure 2: Cactus plot of LTMOA*'s performance versus its virtual best variant, parallelized variant with and without upper bounding (denoted noub). Instances are sorted by runtime.

iii) achieve a speed-up proportional to the problem dimension. Comparing the average speed-up factors across the scenarios, we observe a steady increase as the number of objectives rises, with the average factor growing from 1.3 in the $k = 3$ scenario to around 4 in the $k = 5$ scenario for both algorithms;
iv) require more space to accommodate the scaled search space. Memory usage increases proportionally as well. The parallelized LTMOA* consumes 4.4 times more memory in the $k = 5$ scenario, but only 1.7 times in the $k = 3$ scenario.

To better illustrate the runtime improvement achieved by parallelization, we present in Figure 1 the runtime distribution of each parallel algorithm against the base variant with $k = 4$. We observe that, in almost every individual instance, both LTMOA* and NWMOA* have performed faster when parallelized with our framework, reducing the computation time by up to an order of magnitude.

Impact of upper bounding: To study the impact of our proposed upper-bounding approach on achieving an accelerated search, and also to compare the parallel framework with other variants of the base algorithms, we evaluated LTMOA* over the instances with four objectives, in two other configurations: i) parallelized LTMOA* but with the upper bounding switched off; ii) a virtual best oracle of LTMOA*, which delivers the best runtime among (four) cyclic permutations of objectives. To set up this virtual best oracle, we ran for each instance four runs of the standard LTMOA*, each on a different objective order, and gave the virtual oracle the best runtime. Figure 2 compares the runtime performance of these two variants (denoted by LTMOA*$^{\text{noub}}_{\text{par}}$ and LTMOA*$_{\text{best}}$) against the standard and parallel variants. There are two key observations that underscore the importance of upper bounding. First, the parallel variant outperforms all other variants, including the virtual best variant. Second, although parallelized search with disabled upper bounding can still outperform the standard variant, it falls short of achieving the performance of the virtual best variant. This is primarily due to the considerable search overhead inherent in parallelization. Thus, we can conclude that the proposed upper-bounding technique plays a crucial role in enabling the efficient parallelization of MOA*.

We also evaluated for NWMOA* a (non-parallel) sequential variant of our framework, where the algorithm alternates between the search orders upon discovering a solution in one ordering. This variant offered only marginal improvements over the standard version, underscoring that the framework is more suited to scenarios where parallelization is feasible.

Conclusion and Future Work

This research proposed the first parallel search framework for the multi-objective shortest path problem on the basis of A* search (MOA*). While being applicable to existing MOA* approaches, our framework allows the space of the problem to be searched with different objective orderings simultaneously. The research also builds a unique information-sharing pipeline between concurrent searches that can reduce the dimensionality of the problem to *one* in certain cases. Two of the leading MOA* algorithms were parallelized based on the proposed approach. The results demonstrate the success of the proposed framework in reducing the computation time of MOA*, achieving an average speed-up proportional to the problem dimension.

Future work: The number of threads in our proposed parallel framework is determined by the number of objectives. However, if there are more threads available, the framework can be extended to utilize some of them through incorporating a different set of objective orderings. In this scenario, MOA* in some threads are guided by the same primary cost, leading to significant overlap in their search space. Future research could explore mechanisms that leverage mutual information between these threads to enhance upper bounding.

Acknowledgments

This research was supported by the Department of Climate Change, Energy, the Environment and Water under the International Clean Innovation Researcher Networks (ICIRN) program grant number ICIRN000077. Mahdi Jalili is supported by Australian Research Council through projects DP240100963, DP240100830, LP230100439 and IM240100042.

References

Ahmadi, S.; Sturtevant, N. R.; Harabor, D.; and Jalili, M. 2024a. Exact Multi-objective Path Finding with Negative Weights. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, 11–19. AAAI Press.

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2021. Bi-Objective Search with Bi-Directional A*. In *29th Annual European Symposium on Algorithms, ESA 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, 3:1–3:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2022. Weight Constrained Path Finding with Bidirectional A. In *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria, July 21-23, 2022*, 2–10. AAAI Press.

Ahmadi, S.; Tack, G.; Harabor, D.; Kilby, P.; and Jalili, M. 2024b. Enhanced methods for the weight constrained shortest path problem. *Networks*, 84(1): 3–30.

Erb, S.; Kobitzsch, M.; and Sanders, P. 2014. Parallel Bi-objective Shortest Paths Using Weight-Balanced B-trees with Bulk Updates. In *Experimental Algorithms - 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29 - July 1, 2014. Proceedings*, volume 8504 of *Lecture Notes in Computer Science*, 111–122. Springer.

Erkut, E.; Tjandra, S. A.; and Verter, V. 2007. Hazardous materials transportation. *Handbooks in operations research and management science*, 14: 539–621.

Fukunaga, A.; Botea, A.; Jinnai, Y.; and Kishimoto, A. 2018. Parallel A* for State-Space Search. In *Handbook of Parallel Constraint Reasoning*, 419–455. Springer.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2): 100–107.

Hernández, C.; Yeoh, W.; Baier, J. A.; Felner, A.; Salzman, O.; Zhang, H.; Chan, S.-H.; and Koenig, S. 2023. Multi-objective search via lazy and efficient dominance checks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 7223–7230.

Howe, A. E.; and Dahlman, E. 2002. A Critical Assessment of Benchmark Comparison in Planning. *J. Artif. Intell. Res.*, 17: 1–3.

Knight, K. 1993. Are Many Reactive Agents Better Than a Few Deliberative Ones? In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, 432–437. Morgan Kaufmann.

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

Pulido, F. J.; Mandow, L.; and Pérez-de-la-Cruz, J. 2015. Dimensionality reduction in multiobjective shortest path search. *Comput. Oper. Res.*, 64: 60–70.

Ren, Z.; Rathinam, S.; and Choset, H. 2021. Multi-objective Conflict-based Search for Multi-agent Path Finding. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, 8786–8791. IEEE.

Ren, Z.; Zhan, R.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022. Enhanced Multi-Objective A* Using Balanced Binary Search Trees. In *Proceedings of the Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria*, 162–170. AAAI Press.

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

Sanders, P.; and Mandow, L. 2013. Parallel Label-Setting Multi-objective Shortest Path Search. In *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, 215–224. IEEE Computer Society.

Stewart, B. S.; and White III, C. C. 1991. Multiobjective A*. *J. ACM*, 38(4): 775–814.

Tezcaner, D.; and Köksalan, M. 2011. An Interactive Algorithm for Multi-objective Route Planning. *J. Optim. Theory Appl.*, 150(2): 379–394.

Ulloa, C. H.; Zhang, H.; Koenig, S.; Felner, A.; and Salzman, O. 2024. Efficient Set Dominance Checks in Multi-Objective Shortest-Path Algorithms via Vectorized Operations. In *Seventeenth International Symposium on Combinatorial Search, SOCS 2024, Kananaskis, Alberta, Canada, June 6-8, 2024*, 208–212. AAAI Press.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously Searching with Multiple Settings: An Alternative to Parameter Tuning for Suboptimal Single-Agent Search Algorithms. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 177–184. AAAI.

Weiszer, M.; Burke, E. K.; and Chen, J. 2020. Multi-objective routing and scheduling for airport ground movement. *Transportation Research Part C: Emerging Technologies*, 119: 102734.

Zhou, Y.; and Zeng, J. 2015. Massively Parallel A* Search on a GPU. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, 1248–1255. AAAI Press.