Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-26)
Sister Conferences Best Papers Track

Parallelizing Multi-Objective A* Search (Extended Abstract)*

Saman Ahmadi[1] , **Nathan R. Sturtevant**[2] , **Andrea Raith**[3] , **Daniel Harabor**[1] , **Mahdi Jalili**[4]
[1]Department of Data Science and AI, Monash University, Australia
[2]Department of Computing Science, University of Alberta, Canada
[3]Department of Engineering Science, University of Auckland, New Zealand
[4]School of Engineering, RMIT University, Australia
saman.ahmadi@moansh.edu, 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 strong performance on challenging MOSP instances. This paper presents a novel search framework that enables efficient parallelization of MOA* through different objective orderings and a unique upper-bounding strategy that, in certain cases, allows the problem dimensionality to be reduced to one. Results demonstrate that the proposed framework can significantly improve the performance of MOA*, with speedups increasing proportionally to the number of objectives.

1 Introduction

The Multi-Objective Shortest Path problem (MOSP) 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. Examples are multi-objective path planing for mobile robots [Ren *et al.*, 2021] and multi-objective route selection problem for unmanned air vehicles [Tezcaner and Köksalan, 2011]. Among existing MOSP solution methods, MOA* [Stewart and White III, 1991] is considered state of the art, with recent extensions demonstrating the significant contribution of heuristic search to improving MOA*'s performance in challenging cases [Salzman *et al.*, 2023]. The EMOA* [Ren *et al.*, 2022], TMDA [Maristany de las Casas *et al.*, 2023], LTMOA* [Hernández *et al.*, 2023], and NWMOA* [Ahmadi *et al.*, 2024a] are recent approaches that utilize best-first search to solve point-to-point MOSP more efficiently, with NWMOA* demonstrating better performance than other approaches.

In many domains, the performance of planning algorithms can be impacted by slight modifications to the search setting, such as changes in tie-breaking rules or the ordering of operators [Knight, 1993; Howe and Dahlman, 2002; Ahmadi *et al.*, 2024b]. In the case of multi-objective search, while MOA* provides a robust framework for solving MOSP optimally, the ordering of objectives can have a "*dramatic effect on the algorithm's running time*" [Salzman *et al.*, 2023]. 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 multi-objective search literature. While existing approaches mainly focus on parallelizing search procedures [Sanders and Mandow, 2013; Erb *et al.*, 2014; Ulloa *et al.*, 2024], the bi-objective search algorithm BOBA* [Ahmadi *et al.*, 2021] and its weight-constrained adaptation [Ahmadi *et al.*, 2022] provide a parallel framework that executes two concurrent searches with different objective orderings, but they are limited to handling only two objectives.

This research proposes a parallel search framework for MOA*, representing the first attempt to parallelize multi-objective search beyond two objectives. Inspired by the search scheme of BOBA*, the proposed approach runs multiple MOA* searches in parallel, each operating under a different objective ordering. Central to the framework is an innovative upper-bounding technique that not only shrinks the scaled search space but also reduces the problem dimensionality to one under specific conditions. Experimental results demonstrate the effectiveness of the proposed framework in achieving scalable acceleration of MOA*, with speedups proportional to the number of objectives.

2 Notation and Problem Formulation

We consider a directed graph $G = (S, E)$, where S is a finite set of states and $E \subseteq S \times S$ is a set of directed edges. Each edge $e \in E$ is associated with a vector-valued cost function $\mathbf{cost} : E \to \mathbb{R}^k$, where $\mathbf{cost}(e) = (cost_1, cost_2, \ldots, cost_k)$. A path π is defined as a sequence of states $\langle u_1, u_2, \ldots, u_n \rangle$ such that $(u_i, u_{i+1}) \in E$ for all $i < n$. The cost of a path is obtained by aggregating edge costs componentwise: $\mathbf{cost}(\pi) = \sum_{i=1}^{n-1} \mathbf{cost}(u_i, u_{i+1})$. The MOSP problem aims to find a set of Pareto-optimal paths between a given pair of $(start, goal)$ states. A path π^* is Pareto-optimal if there is no other path π' that simultaneously satisfies $cost_i(\pi') < cost_i(\pi^*)$ and $cost_j(\pi') \leq cost_j(\pi^*)$ for any $i, j \in \{1, \ldots, k\} | i \neq j$.

*Short version of [Ahmadi *et al.*, 2025], ICAPS 2025 Best Paper

In a search setting, we represent paths using nodes. A node x encodes a path ending in state $s(x)$, where $s(\cdot)$ maps nodes to states. Each node maintains (i) a cost vector $\mathbf{g}(x)$ representing the cost from $start$ to $s(x)$, (ii) an estimated total cost vector $\mathbf{f}(x)$ from $start$ to $goal$ via x, and (iii) a pointer $parent(x)$ indicating its predecessor in the search tree. All vector operations are defined element-wise, and vector comparisons use standard ordering, where $\mathbf{f}(x) \preceq \mathbf{f}(y)$ indicates $f_i(x) \leq f_i(y)$ for all $i \in 1, \ldots, k$. We additionally define a truncation operator $\mathrm{Tr}(\mathbf{v})$ that removes the first component of a vector, with Tr^λ meaning λ successive truncations.

Definition. A vector $\mathbf{v}$ is weakly dominated by $\mathbf{v}'$ if $\mathbf{v}' \preceq \mathbf{v}$. It is strictly dominated if $\mathbf{v}' \preceq \mathbf{v}$ and $\mathbf{v}' \neq \mathbf{v}$. A node y is weakly dominated by node x if $\mathbf{f}(x) \preceq \mathbf{f}(y)$.

3 Multi-Objective A* Search

Multi-objective pathfinding with A* involves a systematic search by *expanding* nodes in best-first order, where paths with the smallest estimated $start$-$goal$ costs are given higher priority. These estimates are traditionally established as $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}(s(x))$ where a consistent heuristic function $\mathbf{h} : S \to \mathbb{R}^k$ estimates lower bounds on the $\mathbf{cost}$ of extended paths to $goal$ [Hart *et al.*, 1968].

MOA* maintains a frontier set $Open$ of nodes ordered by $\mathbf{f}$-values, and a solution set $Sols$ containing all non-dominated goal-reaching nodes. For each state $u \in S$, a set $\mathrm{G}^{\mathrm{Tr}}(u)$ stores mutually non-dominated truncated cost vectors corresponding to previously expanded nodes with u. At each iteration, a node x with minimal $\mathbf{f}$-value is extracted from $Open$. The node is discarded if it is weakly dominated either by a previous expansion with $s(x)$ or by already discovered solutions. These conditions ensure that only promising nodes are expanded. To perform dominance tests more efficiently, MOA* exploits a structural property of the best-first search by removing the first cost component, which is known to be non-decreasing. Dominance checks are then performed in this reduced space: a node x is pruned if $\mathrm{Tr}(\mathbf{g}(x))$ (resp. $\mathrm{Tr}(\mathbf{f}(x))$) is weakly dominated by any truncated cost vector in $\mathrm{G}^{\mathrm{Tr}}(s(x))$ (resp. $\mathrm{G}^{\mathrm{Tr}}(goal)$). If x is not pruned, its truncated vector is inserted into $\mathrm{G}^{\mathrm{Tr}}(s(x))$, and any dominated entries are removed to maintain a non-dominated set. If $s(x) = goal$, x is added to $Sols$; otherwise, it is expanded by generating successors over $Succ(s(x))$. Successor nodes can be tested for dominance either upon generation or upon extraction from $Open$, with the latter approach preferred in practice [Hernández *et al.*, 2023]. The search terminates when $Open$ is empty, returning $Sols$ as the set of solutions.

4 Parallelized MOA*

The idea behind our parallel framework is straightforward: multiple MOA* searches are executed simultaneously under different objective orderings. Algorithm 1 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. The parallel loop can terminate as soon as one thread finishes its search.

Algorithm 1: Parallel MOA*- Higher Level

```
   Input: A MOSP Problem (G, start, goal, k)
   Output: A cost-unique Pareto-optimal solution set
1  Sols ← ∅^k, f̄ ← ∞^k
2  for i ∈ {1, ..., k} do in parallel
3      c ← cost function with cost_i as primary cost
4      h ← heuristic function corresponding to c
5      Parallelized MOA* on (G, c, h, start, goal, Sols_i)
6  return Unique(⋃_{i=1}^{k} Sols_i)
```

Upper-bounding mechanism: Naive parallelization may lead to substantial overlap between the search spaces of different threads, resulting in many duplicate solutions. To mitigate this overhead, we design a novel upper-bounding mechanism that reduces the search effort in each parallelized MOA* instance by establishing an information-sharing pipeline between 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 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 solutions obtained in the second thread with $cost_2$ smaller than f'_2 (second thread explores nodes in non-decreasing order of $cost_2$ estimates). The doubly truncated upper bound $(f_3, \ldots, f_k)$ can be further truncated with a similar reasoning: if we observe f_3 within the explored range of the third thread (guided by $cost_3$), 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 $(f_1, \ldots, f_k)$ yields f_k as a scalar upper bound on the k-th objective, reducing the dimension to *one*.

Cost vectors truncated $k-1$ times can serve as *global* upper bounds shared across all threads, while also enabling early termination in parallelized MOA*. Let $\bar{\mathbf{f}} = (\overline{f_1}, \ldots, \overline{f_k})$ denote a vector of scalar global upper bounds, initially set to infinity. Under cyclic permutations of objective orderings, each thread is responsible for optimizing a different primary objective and can therefore establish one of the scalar upper bounds in $\bar{\mathbf{f}}$. These bounds can then be used to prune nodes in any thread whose f-values violate at least one global upper bound. Moreover, the k-th thread (and consequently the entire parallel execution) can terminate early when it extracts a node x with $f_k(x) > \overline{f_k}$, since nodes in that thread are expanded in non-decreasing order of their f_k-values.

Algorithm 2: i-th Parallelized MOA* (guided by f_i)

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** $f_i(x) \not\leq \overline{f_i}$ **then break**
9 **if** IsDominated$(\mathrm{Tr}(\mathbf{g}(x)), \mathrm{G}_1^{\mathrm{Tr}}(s(x)))$ **or** IsDominated$_{\mathrm{MD}}(\mathbf{f}(x), \mathbf{G^{Tr}}(goal))$ **then**
10 **continue**
11 RemoveDominated$(\mathrm{Tr}(\mathbf{g}(x)), \mathrm{G}_1^{\mathrm{Tr}}(s(x)))$
12 Add $\mathrm{Tr}(\mathbf{g}(x))$ to $\mathrm{G}_1^{\mathrm{Tr}}(s(x))$
13 **if** $s(x) = goal$ **then**
14 UpdateUpperBound$(\mathbf{G^{Tr}}(goal))$
15 Add x to $Sols$
16 **continue**
17 **for each** $t \in Succ(s(x))$ **do**
18 $y \leftarrow$ new node with $s(y) = t$
19 $\mathbf{g}(y) \leftarrow \mathbf{g}(x) + \mathbf{cost}(s(x), t)$
20 $\mathbf{f}(y) \leftarrow \mathbf{g}(y) + \mathbf{h}(t)$
21 $parent(y) \leftarrow x$
22 **if** IsDominated$(\mathrm{Tr}(\mathbf{g}(y)), \mathrm{G}_1^{\mathrm{Tr}}(t))$ **or** IsDominated$_{\mathrm{MD}}(\mathbf{f}(y), \mathbf{G^{Tr}}(goal))$ **then**
23 **continue**
24 Add y to $Open$
25 **return**

Algorithm 3: IsDominated$_{\mathrm{MD}}$

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

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

To apply upper bounding, the framework maintains two shared sets: a solution set $\mathbf{Sols} = \{Sols_1, \ldots, Sols_k\}$ together with a global upper-bound vector $\overline{\mathbf{f}}$ (line 1 of Algorithm 1). Algorithm 2 presents the detailed procedure for the i-th parallelized MOA* search guided by f_i. Compared to standard MOA*, the framework introduces three main modifications. First, to support multidimensional truncation, $\mathbf{G^{Tr}}$ now organizes truncated cost vectors according to their truncation level, i.e., $\mathbf{G^{Tr}} = \{\mathrm{G}_1^{\mathrm{Tr}}, \ldots, \mathrm{G}_{\mathrm{k-1}}^{\mathrm{Tr}}\}$, where $\mathrm{G}_{\lambda}^{\mathrm{Tr}}$ stores vectors truncated λ times. Second, dominance checking against previously discovered solutions is generalized through the IsDominated$_{\mathrm{MD}}$ procedure (lines 9, 22). As shown in Algorithm 3, a cost vector $\mathbf{v}$ is first evaluated against the shared upper bound $\overline{\mathbf{f}}$. If it satisfies the bound, the procedure performs a sequence of dominance tests over progressively truncated representations of $\mathbf{v}$. At truncation level λ, the resulting vector is compared against the corresponding set $\mathrm{G}_{\lambda}^{\mathrm{Tr}}(goal)$. Third, whenever a non-dominated solution node is extracted, the framework attempts to tighten the shared upper bounds through the UpdateUpperBound procedure (line 14). This procedure is detailed in Algorithm 4. 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 j 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 j-th thread from $Sols_j$ (or alternatively, the most recently extracted node in the j-th thread). The vector $\mathbf{v}$ can be further truncated if we observe that its f_j-value is no larger than the f_j-value of the retrieved solution (or the recently extracted) node from the j-th thread. The truncation involves: removing the vector from its current set $\mathrm{G}_{\lambda}^{\mathrm{Tr}}$, truncating it, and inserting it into the next set $\mathrm{G}_{\lambda+1}^{\mathrm{Tr}}$, while ensuring the vectors of the set remain non-dominated after the new vector is added. Thus, the last set $\mathrm{G}_{\mathrm{k-1}}^{\mathrm{Tr}}$ 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 $\mathrm{G}_{\lambda+1}^{\mathrm{Tr}}$. If the new vector is dominated, it does not need to be added to the set (line 8 of Algorithm 4). 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 $\mathrm{G}_{\lambda+1}^{\mathrm{Tr}}$ all its vectors weakly dominated by the new vector (line 10). The newly truncated vector will be added into $\mathrm{G}_{\lambda+1}^{\mathrm{Tr}}$ if it is deemed non-dominated (line 11). In the end, if $\mathrm{G}_{\mathrm{k-1}}^{\mathrm{Tr}}$ 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).

Algorithm 4: UpdateUpperBound

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

1 **for** $\lambda \in \{1, \ldots, k-2\}$ **do**
2 $j \leftarrow$ index of the primary cost of vectors in V_{λ}
3 **if** $Sols_j = \emptyset$ **then continue**
4 $z \leftarrow$ The most recent solution node in $Sols_j$
5 **for** $\mathbf{v} \in \mathrm{V}_{\lambda}$ **do**
6 **if** $f_j(\mathbf{v}) \leq f_j(z)$ **then**
7 Remove $\mathbf{v}$ from V_{λ}
8 **if** IsDominated$(\mathrm{Tr}(\mathbf{v}), \mathrm{V}_{\lambda+1})$ **then**
9 **continue**
10 RemoveDominated$(\mathrm{Tr}(\mathbf{v}), \mathrm{V}_{\lambda+1})$
11 Add $\mathrm{Tr}(\mathbf{v})$ to $\mathrm{V}_{\lambda+1}$
12 **if** $\mathrm{V}_{k-1} \neq \emptyset$ **then**
13 $j \leftarrow$ index of the scalar cost in V_{k-1}
14 $\overline{f_j} \leftarrow$ the scalar cost in V_{k-1}
15 **return**

Merging Solutions. Since nodes are not rigorously checked against all solutions in $\mathbf{Sols}$, threads may still generate duplicate solutions. Therefore, the final step is to merge and return the cost-unique solutions (line 6 of Algorithm 1).

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 1: Performance of algorithms over 100 instances in three scenarios. Runtimes are in seconds and $|\mathcal{S}|$ is the number of solved cases. We report, for mutually solved cases, the average memory usage (in GB) and speed up achieved w.r.t. the non-parallel variant.

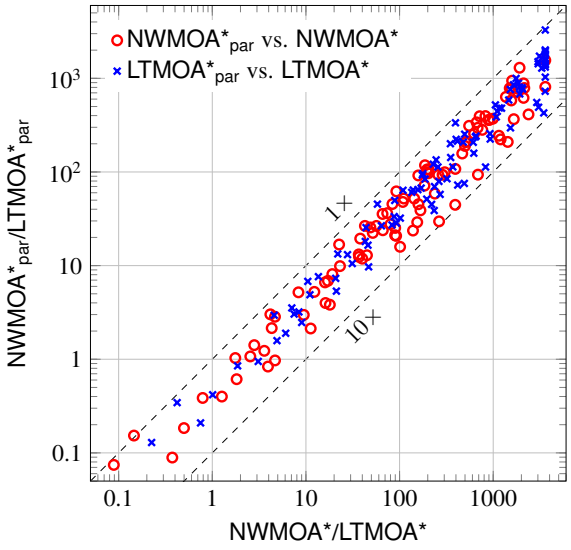

Figure 1: Runtime distribution of NWMOA* and LTMOA* versus their parallelized variant over instances with $k = 4$.

5 Experimental Results

We parallelized two recent MOA* algorithms using our framework: NWMOA*[Ahmadi *et al.*, 2024a] and lazy LTMOA* [Hernández *et al.*, 2023], all implemented in C++. All 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) running at 2.4 GHz and with 16 GB of RAM with a one-hour timeout. Objective orderings in the parallel algorithms were defined using cyclic permutations of objectives, with one thread assigned to each ordering. For the benchmark map, we used the New York map from the 9th DIMACS Implementation Challenge: Shortest Paths (http://www.diag.uniroma1.it/~challenge9/download.shtml) in scenarios with three to five cost components. The edge costs are: (1) *distance*; (2) *time*; (3) the positive height difference between the endpoints of the edge; (4) the average (out)degree of the endpoints; and (5) a unit cost (one). We then evaluated all algorithms on 100 random ($start$, $goal$) pairs per scenario.

Performance Impact of Parallelization. Table 1 compares the performance of our parallelized algorithms (main search) 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;
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;
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, reducing the computation time by up to an order of magnitude. Further, our detailed results demonstrate that parallel LTMOA* outperforms LTMOA* using the best objective ordering, whereas this improvement is not achieved without the proposed upper-bounding technique, highlighting its crucial role in enabling efficient parallelization of MOA*.

6 Conclusion

This research proposed the first parallel search framework for the multi-objective A* search (MOA*). While being applicable to existing MOA*-based 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.

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. Saman Ahmadi is supported by Australian Research Council through project IE250100126, and Mahdi Jalili through projects DP240100963, IC240100010, IM240100042 and LP230100439.

References

[Ahmadi *et al.*, 2021] Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Bi-objective search with bi-directional A*. In *29th Annual European Symposium on Algorithms, ESA 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 3:1–3:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[Ahmadi *et al.*, 2022] Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. 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*, pages 2–10. AAAI Press, 2022.

[Ahmadi *et al.*, 2024a] Saman Ahmadi, Nathan R. Sturtevant, Daniel Harabor, and Mahdi Jalili. 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*, pages 11–19. AAAI Press, 2024.

[Ahmadi *et al.*, 2024b] Saman Ahmadi, Guido Tack, Daniel Harabor, Philip Kilby, and Mahdi Jalili. Enhanced methods for the weight constrained shortest path problem. *Networks*, 84(1):3–30, 2024.

[Ahmadi *et al.*, 2025] Saman Ahmadi, Nathan R. Sturtevant, Andrea Raith, Daniel Harabor, and Mahdi Jalili. Parallelizing multi-objective A* search. *CoRR*, abs/2503.10075, 2025.

[Erb *et al.*, 2014] Stephan Erb, Moritz Kobitzsch, and Peter Sanders. 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*, pages 111–122. Springer, 2014.

[Fukunaga *et al.*, 2018] Alex Fukunaga, Adi Botea, Yuu Jinnai, and Akihiro Kishimoto. Parallel A* for state-space search. In *Handbook of Parallel Constraint Reasoning*, pages 419–455. Springer, 2018.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968.

[Hernández *et al.*, 2023] Carlos Hernández, William Yeoh, Jorge A Baier, Ariel Felner, Oren Salzman, Han Zhang, Shao-Hung Chan, and Sven Koenig. Multi-objective search via lazy and efficient dominance checks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 7223–7230, 2023.

[Howe and Dahlman, 2002] Adele E. Howe and Eric Dahlman. A critical assessment of benchmark comparison in planning. *J. Artif. Intell. Res.*, 17:1–3, 2002.

[Knight, 1993] Kevin Knight. 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*, pages 432–437. Morgan Kaufmann, 1993.

[Maristany de las Casas *et al.*, 2023] Pedro Maristany de las Casas, Luitgard Kraus, Antonio Sedeño-Noda, and Ralf Borndörfer. Targeted multiobjective dijkstra algorithm. *Networks*, 82(3):277–298, 2023.

[Ren *et al.*, 2021] Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. 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*, pages 8786–8791. IEEE, 2021.

[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 Fifteenth International Symposium on Combinatorial Search, SOCS 2022, Vienna, Austria*, pages 162–170. AAAI Press, 2022.

[Salzman *et al.*, 2023] Oren Salzman, Ariel Felner, Carlos Hernández, Han Zhang, Shao-Hung Chan, and Sven Koenig. 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*, pages 6759–6768. ijcai.org, 2023.

[Sanders and Mandow, 2013] Peter Sanders and Lawrence Mandow. 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*, pages 215–224. IEEE Computer Society, 2013.

[Stewart and White III, 1991] Bradley S. Stewart and Chelsea C. White III. Multiobjective A*. *J. ACM*, 38(4):775–814, 1991.

[Tezcaner and Köksalan, 2011] Diclehan Tezcaner and Murat Köksalan. An interactive algorithm for multi-objective route planning. *J. Optim. Theory Appl.*, 150(2):379–394, 2011.

[Ulloa *et al.*, 2024] Carlos Hernández Ulloa, Han Zhang, Sven Koenig, Ariel Felner, and Oren Salzman. 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*, pages 208–212. AAAI Press, 2024.

[Valenzano *et al.*, 2010] Richard Anthony Valenzano, Nathan R. Sturtevant, Jonathan Schaeffer, Karen Buro, and Akihiro Kishimoto. 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*, pages 177–184. AAAI, 2010.

[Zhou and Zeng, 2015] Yichao Zhou and Jianyang Zeng. 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*, pages 1248–1255. AAAI Press, 2015.