SOPMOA*: Unleashing Shared-Open Parallelism for High-Performance Multi-Objective Pathfinding

Long Viet Truong[0009−0002−2213−4184], Tien Minh Dam[0009−0006−7683−4476], Tuan Anh Nguyen, Linh Thuy Thi Nguyen, and Duong Trung Dinh

Viettel High Technology Industries Corporation
{longtv28, tiendm9, tuanna63, linhntt6, duongdt2}@viettel.com.vn

Abstract. The Multi-Objective Shortest Path (MOSP) problem generalizes the classic shortest path problem by simultaneously optimizing multiple, often conflicting, cost functions. Recent advances in MOSP have yielded algorithms that employ sophisticated heuristic-based techniques and dimensionality reduction to expedite search. However, most existing methods rely on strictly sequential frameworks, leaving parallelized approaches relatively underexplored - especially for high-dimensional objectives. In this paper, we introduce SOPMOA* (Shared-Open Parallelized Multi-Objective A*), an algorithm that addresses this gap by enabling any number of concurrent sub-searchers to cooperate via a shared-memory priority queue. Each sub-searcher independently processes labels, performs dominance checks against locally stored partial Pareto fronts, and contributes to a global frontier of non-dominated solutions. We propose mechanisms for safe and efficient updates to shared data structures, ensuring correctness without excessive locking overhead. Empirical evaluations on benchmark multi-objective road networks demonstrate that SOPMOA* scales favorably with increasing parallelism and consistently outperforms state-of-the-art algorithms such as EMOA*, LTMOA*, and NWMOA* in both speed and robustness. These results underscore the substantial potential of shared-memory parallelization in tackling challenging multi-objective pathfinding tasks.

Keywords: Multi-Objective Shortest Path (MOSP) · Parallel A* Search · Shared-Memory Parallelism · Heuristic Search · Concurrent Algorithms · Multi-Objective Optimization.

1 Introduction

1.1 Problem Definition

Consider a directed graph $G = \langle V, E, c \rangle$, where V is a finite vertex set, E is the set of directed edges (u, v) with $u \neq v$, and $c : E \to \mathbb{R}^d_{\geq 0}$ is a non-negative cost function with d attributes. A path p is a sequence of vertices v_i ($i \in \overline{1..n}$) such that $(v_i, v_{i+1}) \in E$. The total cost of p is $c(p) = \sum_{i=1}^{n-1} c(v_i, v_{i+1})$.

A cost vector α *weakly dominates* β ($\alpha \preceq \beta$) if $\alpha_i \leq \beta_i$ for all i; α *dominates* β ($\alpha \prec \beta$) if $\alpha \preceq \beta$ and $\alpha \neq \beta$. If neither dominates the other, they are *mutually undominated*. The vector α is *lexicographically smaller* than β ($\alpha <_{lex} \beta$), if the first k indices $i \in \overline{1..k}$ satisfy $\alpha_i = \beta_i$, and at index $k+1$, $\alpha_{k+1} < \beta_{k+1}$. The truncated version of $\alpha \in \mathbb{R}^d$, denoted $Tr(\alpha)$, is the $(d-1)$-dimensional vector obtained by removing its first component (i.e., α from the second to last index).

In a search problem with start node s and target t, a heuristic $h : V \to \mathbb{R}^d_{\geq 0}$ estimates the cost from a node to t. In the scope of this paper, h is consistent and admissible in each cost attribute. A path p_{sv} from s to t is represented by a label l with $node(l) = v$, $g(l) = c(p_{sv})$, and $f(l) = g(l) + h(v)$. The Pareto front $\mathcal{P}_v$ at node v is the set of mutually undominated labels among all paths from s to v. The Multi-Objective Shortest Path (MOSP) problem seeks the Pareto front $\mathcal{P}_t$ at the target node, representing all optimal trade-off solutions.

1.2 Algorithmic Background

In recent years, the Multi-Objective Shortest Path (MOSP) problem has attracted increasing attention in network optimization. Heuristic search methods for obtaining the full Pareto set utilize a priority queue of labels: each iteration pops the most promising label, applies a "dominance check" against expanded non-dominated labels, expands and generates successors if qualified. The algorithms terminate when the queue is empty.

The concept of "dimensionality reduction", introduced in NAMOA*$_{dr}$ [8], decreases the size of the non-dominated set, significantly reducing the dominance-check runtime. NAMOA*$_{dr}$'s Eager Check maintains mutually non-dominated labels per node in $OPEN$, guaranteeing that popped labels are already Pareto-optimal, but incurs overhead removing dominated intermediates and re-structuring queue, which reduces queue efficiency. By contrast, LTMOA* [6] and EMOA* [9] employ Lazy Check: $OPEN$ may hold dominated labels from the same node, so each popped label must be checked against the Pareto front before expansion. Both share the same framework and utilize dimensionality reduction; EMOA* stores the truncated front in an AVL tree, while LTMOA* uses a linear list or array. NWMOA* [2], originally for negative-weight graphs, also excels on non-negative graphs by using a bucket priority queue, obviating strict lexicographical order, adopting alternative dominance and frontier-update strategies.

Parallelism in MOSP remains under-explored: early parallel algorithms (e.g. [10, 7]) are often complex or yield modest gains. BOBA* [3] introduces a bi-objective bidirectional search - one forward from the start, one backward from the target - each prioritizing a different objective. Extending this to d objectives, [1] assigns up to d workers to cyclic permutations of objectives, each exploring a distinct region of the Pareto front, though parallelism remains bounded by d. Other approaches, such as BDA [11] and MDA [4], parallelize subtasks like dominance checks and $OPEN$ updates. T-MDA [5] further incorporates BOBA*-style bidirectionality for bi-objective search.

Algorithm 1: SOPMOA* High-level

Input: graph $G\langle V, E, c\rangle$, heuristic h, start s, target t, num. workers N
Output: Complete Pareto optimal solutions $\mathcal{P}_t$

```
1 OPEN ← ∅; Sols ← ∅; G_cl[v] ← ∅, ∀v ∈ V; active[i] ← true, ∀i ∈ 1..N;
2 Initialize start_label of node s; g(start_label) ← 0⃗; f(start_label) ← h(s);
3 Push start_label to OPEN;
4 for id ← 1 to N do in parallel SubSeacher(id);
5 Remove dominated solutions from Sols;
6 return Sols;
```

2 Shared-Open Parallelized Multi-Objective A*

This section introduces the Shared-Open Parallelized Multi-Objective A* (SOPMOA*) algorithm. The algorithm begins by initializing shared-memory components:

- Priority queue $OPEN$: A priority queue storing generated labels in lexicographically ascending f values order.
- List of Pareto fronts G_{cl}: Stores cost vectors g of expanded labels belonging to the truncated Pareto fronts of each node. Lexicographical order is maintained on full form of the costs, while dominance checks and frontier update use dimensionality reduction.
- Solution set $Sols$: Stores qualified labels of target node t, used as algorithm's output.
- Array $active$: Tracks worker status using boolean values, enabling workers to update their own status and monitor others.

A start label is inserted into $OPEN$, then SOPMOA* creates N parallel workers with each act as a sub-searcher (Algorithm 1). These sub-searchers share the workload by simultaneously take labels from $OPEN$ and expand the labels to the shared search tree (represented by G_{cl}). A sub-searcher of SOPMOA* follows the baseline of multi-objective search framework, is shown in Algorithm 2.

The notable difference of SOPMOA* sub-searchers from other algorithms is that they utilize memory-locking mechanism for managing shared attributes. When a sub-searcher detects that $OPEN$ is empty, it does not terminate but instead monitors the status of other sub-searchers, anticipating new labels generated by others. The terminate condition is met when all workers are inactive, indicating that no new label will be generated and the $OPEN$ set is truly empty.

In this parallelism setting, labels may not be processed in strict lexicographical order, unlike in synchronized algorithms. To handle potential dimensionality reduction dominance check issues, we use specialized frontier check and update procedures:

- **Frontier Check**: Given a node v, label x, or target node t with cost vector α (i.e., $g(x)$ or $f(x)$), the frontier check compares α against the Pareto

Algorithm 2: Sub-Searcher

```
   Input: Sub-searcher's index ID
 1 while ∃i ∈ 1..N : active[i] = true do
 2     lock OPEN do
 3         if OPEN is empty then
 4             active[ID] ← false;
 5             continue;
 6         else
 7             Pop label x on top of OPEN;
 8             active[ID] ← true;
 9     if FC(node(x), g(x)) or FC(target, f(x)) then continue;
10     FUpdate(node(x), g(x));
11     if node(x) = target then
12         lock Sols do Add label x to Sols;
13         continue;
14     for succ ∈ successors(node(x)) do
15         Create new label y of node succ;
16         g(y) ← g(x) + c(node(x), succ); f(y) ← g(y) + h(succ);
17         if FC(succ, g(y)) or FC(target, f(y)) then continue;
18         lock OPEN do Push y to OPEN;
```

front $G_{cl}[v]$. A snapshot of $G_{cl}[v]$ is taken under a shared-lock allowing concurrent reads but blocking updates. Vectors lexicographically larger than α are excluded, while dominance is checked on the truncated vectors of those lexicographically smaller than or equal to α, as in the synchronized version.

– **Frontier Update**: If a label is undominated by both frontiers, its $\alpha = g(x)$ is inserted into $G_{cl}[v]$ under an exclusive lock. Before insertion, any vector with a truncated version dominated by α is removed. α is then placed to maintain lexicographical order.

Though true Pareto optimal labels cannot be falsely discarded, false Pareto labels, which do not belong to the Pareto front, can still be retained due to non-monotonic expansion order. These labels are not large in number and are suboptimal because they have to qualify the frontier check beforehand. This abundance does not affect the overall result, as will be proven in Lemma 4.

In the end, previously expanded false Pareto labels are removed from $Sols$. The algorithm returns the complete set of Pareto optimal solutions.

3 Theoretical Results

Lemma 1. *Let cost vector α be lexicographically smaller than cost vector β. Then, α is not dominated by β.*

Proof. If β dominates α, then $\forall i \in \overline{1..d} : \beta_i \leq \alpha_i$. Since $\alpha <_{lex} \beta$, there exists at least one index j such that $\alpha_j < \beta_j$, implying that α is not dominated by β.

Algorithm 3: Frontier Check (FC)

Input: a node v, a cost vector α
Output: **true** if α is dominated by any of $G_{cl}[v]$, else **false**

shared-lock G_{cl} at node v **do** $X \leftarrow copy(G_{cl}[v])$;
for $i_{ub} \leftarrow |X|$ **down to** 0 **do**
  if $i_{ub} > 0$ **and** $X[i_{ub}] \prec_{lex} \alpha$ **then break**;
for $j \leftarrow 1$ **to** i_{ub} **do** `// no loop if` $i_{ub} = 0$
  if $Tr(X[j]) \preceq Tr(\alpha)$ **then return true**;
return false;

Algorithm 4: Frontier Update ($FUpdate$)

Input: a node v, a cost vector α
lock G_{cl} at node v **do**
  for $i \leftarrow 1$ **to** $|G_{cl}[v]|$ **do**
    if $Tr(\alpha) \prec Tr(G_{cl}[v][i])$ **then** Delete at index i of $G_{cl}[v]$;
  for $i_{ins} \leftarrow |G_{cl}[v]|$ **down to** 0 **do**
    if $i_{ins} > 0$ **and** $G_{cl}[v][i_{ins}] \prec_{lex} \alpha$ **then break**;
  Insert α to $G_{cl}[v]$ at index $(i_{ins} + 1)$;

Lemma 2. *Let cost vector α lexicographically smaller than cost vector β, then, α weakly dominates β iff $Tr(\alpha)$ weakly dominates $Tr(\beta)$.*

Proof. $\alpha <_{lex} \beta$ implies that $\alpha_1 \leq \beta_1$. For $\alpha \preceq \beta$, all $\alpha_1 \leq \beta_1$, $\alpha_2 \leq \beta_2$, $\alpha_3 \leq \beta_3$, $\ldots$, $\alpha_d \leq \beta_d$ must be satisfied. Since the first inequality is already met, it must hold $\alpha_2 \leq \beta_2$, $\alpha_3 \leq \beta_3$, $\ldots$, $\alpha_d \leq \beta_d$, equivalent to $Tr(\alpha) \preceq Tr(\beta)$.

Lemma 3. *The procedure using dimensionality reduction in SOPMOA* frontier check without lexicographical-ordering guarantee, returns accurate result (i.e. equivalent to the frontier check on full-form vectors).*

Proof. Let α be the cost and X the Pareto front, partitioned as $X = X_{\leq\alpha} \cup X_{>\alpha}$, where $X_{\leq\alpha}$ contains labels lexicographically non-larger than α, and $X_{>\alpha}$ the rest. The labels in $X_{>\alpha}$ cannot dominate α from Lemma 1, thus are discarded from frontier check. A label in $X_{\leq\alpha}$ can be equal to α, which truncated form weakly dominates $Tr(\alpha)$. Otherwise, the truncated comparisons are valid when the labels are truly lexicographically smaller than α (proved in Lemma 2).

Lemma 4. *A true Pareto label will not be discarded in frontier check by the false labels in the Pareto front.*

Proof. Let α be the cost and X the Pareto front, partitioned as $X = X_{\leq\alpha} \cup X_{>\alpha}$, where $X_{\leq\alpha}$ contains labels lexicographically non-larger α, and $X_{>\alpha}$ the rest. False labels in $X_{>\alpha}$ are excluded from comparisons with α. Otherwise, considering a false label $\beta \in X_{\leq\alpha}$, it must hold that $Tr(\beta) \not\preceq Tr(\alpha)$. Since α is

of the Pareto front while β is not, α either dominates β (ruled out by Lemma 1) or is mutually undominated with β and $\alpha \neq \beta$. In the latter case and $\alpha \neq \beta$, there exists indices i and j that $\alpha_i < \beta_i$ and $\alpha_j > \beta_j$. Since $\beta <_{lex} \alpha$, in the first k ($k \geq 1$) indices, α is no smaller than β, forcing $i > k$ (hence $i > 1$). Thus $Tr(\beta) \not\preceq Tr(\alpha)$, and no false label β can affect α.

Corollary 1. *The pareto optimal labels will always be expanded (will not be discard in dominance check).*

Theorem 1. *SOPMOA* computes cost-unique complete pareto optimal solutions.*

Proof. SOPMOA* employs N parallel sub-searchers following the same multi-objective search framework. The process of frontier check in each sub-searchers is guaranteed to be accurate (Lemma 3) and will not discard true Pareto labels of every node (Lemma 4). Consequently, all true Pareto labels are expanded in frontier updates (sequentially at each node, avoiding conflicts) and generate their successors. This ensures the exploration of all optimal labels in the complete search tree, including the target's Pareto front. However, false Pareto labels are also expanded due to non-monotonic lexicographical order, this only creates abundant branches to the search tree, not affecting the result. Those of the target node will be stripped after the parallelized process, thus SOPMOA* returns complete pareto optimal solutions.

4 Experimental Results

In this section, we evaluate SOPMOA* via two experiments. First, we vary the number of workers to assess how performance scales with parallelism. Second, we compare SOPMOA*'s best and worst configurations against state-of-the-art algorithms EMOA* [9], LTMOA* [6], and NWMOA* [2].

All algorithms were implemented in C++ using standard libraries. SOPMOA* employs OpenMP with `std::mutex` and `std::shared_mutex`, and Intel TBB's `concurrent_priority_queue` for $OPEN$. EMOA*, LTMOA*, and NWMOA* (with bucket PQ) were re-implemented; LTMOA*'s array-G_{cl} version was omitted due to unstable memory handling. Experiments ran on a 24-core Intel Xeon Gold 6242R (3.10 GHz, 32 GB RAM) with a 3600s timeout.

Benchmarks use the DIMACS New York map with two base objectives (distance, time), extended to four by adding economic cost, random integers in [1,100]. Heuristics derive from d backward Dijkstra one-to-all searches and are excluded from reported runtimes.

Experiment 1: SOPMOA* was run with 4, 8, 12, 16, and 20 workers on 50 random start–target pairs. Table 1 shows average runtime, generated nodes, and expanded nodes. Runtime drops sharply from 4 to 8 workers (≈113s), then tapers (only 23s from 16 to 20). Generated and expanded labels rise slightly, likely due to parallelism-induced extraction disorder. Overall, SOPMOA* benefits substantially from increased parallelism.

50 random NY instances (3 objectives) - *avg. solutions* ≈ 8930			
SOPMOA*	*avg. runtime* (s)	*avg. generated*	*avg. expanded*
4 workers	521.231	11 908 051	9 517 206
8 workers	408.084	11 915 313	9 519 256
12 workers	334.939	11 920 603	9 520 361
16 workers	274.189	11 922 836	9 520 568
20 workers	250.243	11 923 698	9 521 368

Table 1. The average runtimes, the average number of generated labels, the average number of expanded labels on 100 random NY instance with 3 objectives, run by five configurations of SOPMOA* (4, 8, 12, 16, 20 workers)

Algorithms	*Solved*	*Runtime* (s)			
		Min	*Max*	*Mean*	*Median*
100 random NY instances (3 objectives) - *avg. solutions* ≈ 10220					
EMOA*	100/100	**0.209**	1896.45	357.84	58.36
LTMOA*	99/100	0.313	3600.00	823.95	158.38
LazyLTMOA*	100/100	0.273	3528.92	690.43	131.18
NWMOA*	100/100	0.234	3456.15	553.50	102.41
SOPMOA* - *4 workers*	100/100	0.311	3024.69	563.13	97.60
SOPMOA* - *20 workers*	100/100	0.257	**1730.34**	**248.58**	**51.83**
50 random NY instances (4 objectives) - *avg. solutions* ≈ 16866					
EMOA*	38/50	**0.507**	3600.00	1344.37	852.71
LazyLTMOA*	33/50	0.896	3600.00	1766.10	1804.97
NWMOA*	37/50	0.579	3600.00	1398.89	980.76
SOPMOA* - *20 workers*	**46/50**	0.768	3600.00	**1096.97**	**649.82**

Table 2. The number of solved instances and the runtime statistics on 100 instances with 3 objectives and 50 instances with 4 objectives, run by EMOA*, LTMOA*, LazyLTMOA*, NWMOA* and two versions of SOPMOA* (4 and 20 workers)

Experiment 2: We compared SOPMOA* (4- and 20-worker setups) to EMOA*, LTMOA*, LazyLTMOA*, and NWMOA* on 100 random 3-objective instances. LTMOA* failed one instance; all others solved all 100. The 20-thread SOPMOA* is ≈1.5× faster than EMOA* and more than 3× faster than LTMOA*, while 4-worker SOPMOA* matches NWMOA*. On 50 4-objective instances, 20-thread SOPMOA* solved 46/50 and maintained superior runtimes, though its minimum runtime is higher due to multithreading overhead.

These results confirm that SOPMOA* achieves superior speed and consistency, underscoring the critical role of parallelization in multi-objective search.

5 Conclusions

In conclusion, we introduced SOPMOA*, a novel parallel MOSP algorithm that delivers complete Pareto optimal solutions via multiple simultaneous subsearchers on a shared priority queue. It features unique strategies for expanded

Pareto fronts, dominance checking, and frontier updating, and in its early development has shown substantial improvements over state-of-the-art algorithms, highlighting its potential to significantly enhance performance and leverage parallelism in MOSP.

References

1. Ahmadi, S.: Parallelizing Multi-objective A* Search (Extended Abstract). Proceedings of the International Symposium on Combinatorial Search **17**, 253–254 (Jun 2024). https://doi.org/10.1609/socs.v17i1.31567
2. Ahmadi, S., Sturtevant, N.R., Harabor, D., Jalili, M.: Exact Multi-objective Path Finding with Negative Weights. Proceedings of the International Conference on Automated Planning and Scheduling **34**, 11–19 (May 2024). https://doi.org/10.1609/icaps.v34i1.31455
3. Ahmadi, S., Tack, G., Harabor, D., Kilby, P.: Bi-Objective Search with Bi-Directional A*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPICS.ESA.2021.3, https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ESA.2021.3
4. Maristany de las Casas, P., Sedeño-Noda, A., Borndörfer, R.: An Improved Multiobjective Shortest Path Algorithm. Computers amp; Operations Research **135**, 105424 (Nov 2021). https://doi.org/10.1016/j.cor.2021.105424
5. Casas, Pedro Maristany de las and Kraus, Luitgard and Sedeño-Noda, Antonio and Borndörfer, Ralf: Targeted multiobjective dijkstra algorithm (2021). https://doi.org/10.48550/ARXIV.2110.10978, https://arxiv.org/abs/2110.10978
6. Hernández, C., Yeoh, W., Baier, J.A., Felner, A., Salzman, O., Zhang, H., Chan, S.H., Koenig, S.: Multi-objective Search via Lazy and Efficient Dominance Checks. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. pp. 7223–7230. IJCAI-2023, International Joint Conferences on Artificial Intelligence Organization (Aug 2023). https://doi.org/10.24963/ijcai.2023/850
7. Medrano, F.A., Church, R.L.: A Parallel Computing Framework for Finding the Supported Solutions to a Biobjective Network Optimization Problem. Journal of Multi-Criteria Decision Analysis **22**(5–6), 244–259 (Apr 2015). https://doi.org/10.1002/mcda.1541
8. Pulido, F.J., Mandow, L., Pérez-de-la Cruz, J.L.: Dimensionality reduction in multiobjective shortest path search. Computers amp; Operations Research **64**, 60–70 (Dec 2015). https://doi.org/10.1016/j.cor.2015.05.007
9. Ren, Z., Zhan, R., Rathinam, S., Likhachev, M., Choset, H.: Enhanced Multi-Objective A* Using Balanced Binary Search Trees. Proceedings of the International Symposium on Combinatorial Search **15**(1), 162–170 (Jul 2022). https://doi.org/10.1609/socs.v15i1.21764
10. Sanders, P., Mandow, L.: Parallel Label-Setting Multi-objective Shortest Path Search. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. p. 215–224. IEEE (May 2013). https://doi.org/10.1109/ipdps.2013.89
11. Sedeño-noda, A., Colebrook, M.: A biobjective Dijkstra algorithm. European Journal of Operational Research **276**(1), 106–118 (Jul 2019). https://doi.org/10.1016/j.ejor.2019.01.007