Parallelizing Multi-objective A* Search (Extended Abstract)

Saman Ahmadi
School of Engineering, RMIT University, Australia
saman.ahmadi@rmit.ed.au

Abstract

The Multi-objective Shortest Path (MOSP) problem 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* have demonstrated superior performance in solving difficult MOSP instances. This paper proposes a novel parallel multi-objective search framework that can accelerate recent A*-based solutions by several factors.

Introduction

Given a graph $G = (S, E)$ with a finite set of states S and a set of edges $E \subseteq S \times S$, where each link represents an array of $k \in \mathbb{N}$ attributes in the form of nonnegative **cost** $= (cost_1, cost_2, \ldots, cost_k)$, the Multi-objective Shortest Path problem (MOSP) aims to find a set of cost-unique Pareto-optimal paths between a given pair of $start \in S$ and $goal \in S$, a set in which every individual solution offers a path that minimizes the multi-criteria problem in all dimensions.

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 search efficiency. The EMOA* (Ren et al. 2022) and LTMOA* (Hernández et al. 2023) algorithms are two state-of-the-art solutions that utilize (heuristic-guided) multi-objective A* (MOA*) 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 degrades when the number of objectives increases, leaving some difficult MOSP instances unsolved even after a one-hour runtime.

Although MOA* provides a simple yet powerful framework to optimally solve MOSP in large graphs, it is known that the order of objectives can dramatically affect the execution time, as discussed in Salzman et al. (2023). For example, LTMOA* may exhibit significantly better runtime if its MOA * is performed in $(cost_3, cost_2, cost_1)$ rather than in the conventional lexicographical order for k = 3. Although a good-performing ordering can be obtained empirically, as in (Hernández et al. 2023), there is no guarantee that such ordering performs best in all instances. Despite being under-explored, parallelizing multi-objective search can be seen as a potential solution to the above shortcoming. For the bi-objective variant, the bidirectional search of Ahmadi et al. (2021), known as BOBA*, is an efficient parallel framework designed to build the Pareto front using both possible objective orderings. This research leverages the search scheme of BOBA* and proposes the first parallel framework for MOA* that can effectively improve its runtime by several factors.

Algorithm 1: Parallel Multi-objective A* Search

```
Input: A MOSP Problem ($G$, $start$, $goal$, $k$)
Output: A cost-unique Pareto-optimal solution set
1 $\mathbf{h}(u) \leftarrow$ cost-optimal path from $u$ to $goal$ $\forall u \in S$
2 for $i \in \{1 \ldots k\}$ do in parallel
3     $Sols_i \leftarrow$ MOA* on ($G$, $\mathbf{h}$, $start$, $goal$) guided by $cost_i$
4 return Unique($\sum_{i=1}^{k} Sols_i$)
```

Parallelizing Multi-objective Search

Algorithm 1 presents the high-level description of the proposed parallel MOA*. Similar to BOBA*, parallel searches are commenced once the heuristic function $\mathbf{h}$ is established. Given a k-dimensional MOSP instance, the algorithm runs k individual MOA* searches (e.g., LTMOA*), each guided by one of the costs as the primary objective (line 3). A possible set of orderings can simply be all cyclic permutations of the objectives. For example, the second search in a three-dimensional instance (k = 3) will be conducted on the $(cost_2, cost_3, cost_1)$ order, where MOA* is guided by estimated $cost_2$ of paths as the primary objective. Since each MOA* search is complete, the parallel loop can be exited as soon as one of the MOA* searches is terminated. Each search finds a subset (or potentially all) of Pareto-optimal solutions, so the final task is to merge all solutions (line 4).

Although this technique allows for more than one objective ordering to be involved in the search, it does not necessarily result in enhancing the overall computation time if the searches are conducted independently, primarily due to the significant overhead associated with parallelization. To improve the search efficiency in the parallel setting, we propose a novel mechanism to shrink the search space by utilizing a unique upper bounding strategy via shared solutions.

To reduce the overhead associated with searching the same space, leading to discovering duplicate solutions, each individual MOA* search needs to be informed with the optimal solutions discovered in the other concurrent searches. This allows unpromising paths to be removed if they lead to a *start-goal* path no better than any discovered optimal solution. The standard MOA* search, however, checks paths against the solutions obtained in the current search only. Let Sol contain *all* **cost**-unique solutions obtained during the k (parallelized) searches. Each individual MOA* search can now use Sol to prune some unpromising paths, reducing the search space by forming a larger set of optimal solution paths as global upper bounds. Nonetheless, this method is not efficient in practice, essential because such linear-time upper-bound pruning becomes costly in the absence of a unified objective ordering. To address this shortcoming, this research designs a novel mechanism to inform each search about the progress made by the other concurrent searches. Let $(g_1, g_2, \ldots, g_k)$ be the **cost** of solution x obtained in the first search guided by $cost_1$. We know that MOA* prunes paths showing estimated $(cost_2, \ldots, cost_k)$ no smaller than $(g_2, \ldots, g_k)$ due to the first dimension already being non-decreasing. Also assume that the second search, guided by $cost_2$, has just discovered the solution y with **cost** = $(g'_2, \ldots, g'_k, g'_1)$. With this new solution, the dimension in the upper-bound pruning of the first search can be further reduced if we observe $g'_2 > g_2$, that is, we just need to check the estimated $(cost_3, \ldots, cost_k)$ of paths in the first search against $(g_3, \ldots, g_k)$ with the second dimension also removed. This pruning is correct, as it basically means extension of paths showing estimated costs no smaller than $(g_3, \ldots, g_k)$ – in any dimension – would definitely lead to a *start-goal* path no better than either x or one of the optimal solutions obtained before y in the second search. Note that when the second search finds y, it guarantees that all solutions with $cost_2$ smaller than g_2 are already captured. The same strategy can be applied to other concurrent searches to potentially reduce the dimension in upper-bound pruning to one, enabling fast $O(1)$ dominance check against some (shared) subsets of solutions.

Experimental Results

We implemented our parallel framework based on LTMOA* with lazy dominance tests in C++. For the benchmark, following the literature, we used the New York map from the 9th DIMACS Implementation Challenge: Shortest Paths (http://www.diag.uniroma1.it/ challenge9/download.shtml) and generated 100 random instances with four edge cost components, namely: 1) distance 2) time 3) average outdegree of link endpoints 4) one (unit cost). We ran our experiment on four cores of an Intel Xeon Gold 5220R processor running at 2.2 GHz and with 64 GB of RAM , under the CentOS Linux 7 environment and with a two-hour timeout.

Table 1 compares the performance of our parallelized LTMOA* against its standard version, which uses the lexicographical ordering of objectives, in both runtime and memory aspects. The parallel framework solves more instances and exhibits better runtime statistics. Our detailed results over the mutually solved instances show that the parallel framework performs 3.3 times faster than the standard algorithm, whilst consuming almost two times more memory on average. We also compared in Figure 1 the runtime performance of our parallel framework against LTMOA* guided with four cyclic permutations of objectives. We observe that our parallel approach outperforms all four variants of LTMOA* by solving more instances in a limited time.

		Runtime(s)				Mem.
Method	$\|S\|$	Min.	Mean$_A$	Mean$_G$	Max.	(GB)
Standard	98	0.38	1347.23	284.10	7200.0	1.05
Parallelized	100	0.27	427.17	93.06	4820.3	2.04

Table 1: Performance of LTMOA* over 100 instances. $|S|$ is the number of solved cases, Mean$_A$ and Mean$_G$ are Arithmetic and Geometric mean, respectively, and Mem. shows memory usage (in GB) over mutually solved instances. The runtime of unsolved cases is considered to be two hours.

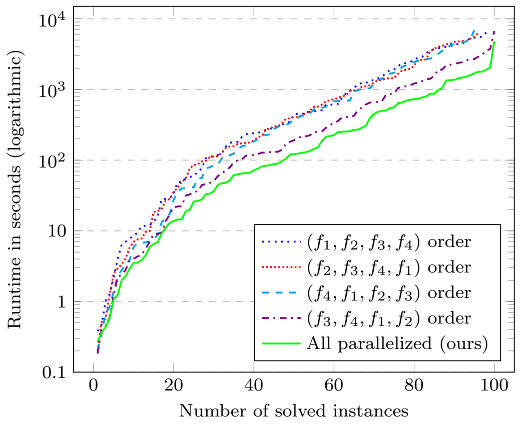

Figure 1: Cactus plot of LTMOA*'s performance in four different orderings of costs versus our parallelized variant.

Acknowledgments

Research supported by Australian Government through the International Clean Innovation Researcher Networks grant.

References

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2021. Bi-Objective Search with Bi-Directional A*. In *ESA*, volume 204 of *LIPIcs*, 3:1–3:15.

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 *IJCAI*, 7223–7230.

Ren, Z.; Zhan, R.; Rathinam, S.; Likhachev, M.; and Choset, H. 2022. Enhanced Multi-Objective A* Using Balanced Binary Search Trees. In *SoCS*, 162–170.

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 *IJCAI*, 6759–6768.