Efficient Set Dominance Checks in Multi-Objective Shortest-Path Algorithms via Vectorized Operations

Carlos Hernández Ulloa[1], **Han Zhang**[2], **Sven Koenig**[2], **Ariel Felner**[3], **Oren Salzman**[4]
[1] Universidad San Sebastián
[2] University of Southern California
[3] Ben-Gurion University
[4] Technion—Israel Institute of Technology
carlos.hernandez@uss.cl, zhan645@usc.edu, skoenig@usc.edu, felner@bgu.ac.il, osalzman@cs.technion.ac.il

Abstract

In the multi-objective shortest-path problem (MOSP) we are interested in finding paths between two vertices of a graph while considering multiple objectives. A key procedure, which dominates the running time of many state-of-the-art (SOTA) algorithms for MOSP is *set dominance checks* (SDC). In SDC, we are given a set X of N-dimensional tuples and a new N-dimensional tuple p and we need to determine whether there exists a tuple $q \in X$ such that q dominates p (i.e., if every element in q is lower or equal than the corresponding element in p). In this work, we offer a simple-yet-effective approach to perform SDC in a parallel manner, an approach that can be seamlessly integrated with most SOTA MOSP algorithms. Specifically, by storing states in memory dimension-wise and not state-wise, we can exploit vectorized operations offered by "Single Instruction/Multiple Data" (SIMD) instructions to efficiently perform SDC on ubiquitous consumer CPUs. Integrating our approach for SDC allows to dramatically improve the runtime of existing MOSP algorithms.

1 Introduction

The *multi-objective shortest-path problem* (MOSP) is a variation of the classical shortest-path problem where the goal is to find paths between two points in a graph while considering multiple objectives instead of a single objective (Ulungu and Teghem 1991; Salzman et al. 2023). Applications of MOSP range from route planning for power lines considering economic and ecological impacts (Bachmann et al. 2018), transporting hazardous materials considering travel distance and risk (Bronfman et al. 2015), and inspecting a region of interest using cameras placed on-board robotic platforms (Fu et al. 2019; Fu, Salzman, and Alterovitz 2021).

In contrast to the single-objective setting where we are typically interested in computing a shortest path between two vertices on a graph $G = (S, E)$, in MOSP we wish to compute the set of *Pareto-optimal* solutions Π^* between two given vertices. Intuitively, Π^* contains all "interesting" paths—the set of solutions for which no other solution can improve one objective without worsening at least one other objective. However, computing Π^* is NP-hard (Serafini 1987) and the cardinality of Π^* may be exponential in $|S|$ (Hansen 1980; Ehrgott 2005; Breugem, Dollevoet, and van den Heuvel 2017).

A key operation in MOSP search algorithms is *set dominance checks* (SDC) which often governs the algorithm's running time (Salzman et al. 2023). Here, we are given a set $\mathbf{X}$ of N-dimensional tuples (with $N \geq 2$) and a new N-dimensional tuple $\mathbf{p}$ and we are interested in testing whether there exists a tuple $\mathbf{q} \in \mathbf{X}$ such that every element in $\mathbf{q}$ is smaller or equal to the corresponding element in $\mathbf{p}$.

To reduce the computational overhead incurred by SDC, existing algorithms employ algorithmic techniques such dimensionality reduction (Pulido, Mandow, and De la Cruz 2015; Hernández et al. 2023) and lazy computation (Hernandez et al. 2023). While successfully reducing the computational cost of SDC, it still remains a key computational bottleneck, especially when the number of objectives grows.

In this work, we offer a complementary approach wherein we perform SDC in a parallel manner, an approach that can be seamlessly integrated with most state-of-the-art MOSP algorithm. Specifically, by storing states in memory dimension-wise and not state-wise, we can exploit vectorized operations offered by "Single Instruction/Multiple Data" (SIMD) instructions to perform SDC with high throughput and low latency on ubiquitous consumer CPUs, accelerating almost any MOSP algorithm. These insights allow us to obtain speedups of up to $8\times$ on challenging planning instances.

2 Preliminaries

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

Boldface font is used to represent N-tuples. We assume an N-tuple $\mathbf{p}$ has the form $\mathbf{p} = (p_1, p_2, \ldots, p_N)$, thus p_i denotes the i^{th} component of an N-tuple. The addition of two N-tuples $\mathbf{p}$ and $\mathbf{q}$ is defined as $\mathbf{p} + \mathbf{q} = (p_1 + q_1, \ldots, p_N + q_N)$. $\mathbf{c}(\pi) = \sum_{i=1}^{m-1} \mathbf{c}(s_i, s_{i+1})$ is the cost of path $\pi = s_1, \ldots, s_m$.

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Given two N-tuples $\mathbf{p}$ and $\mathbf{q}$, we say that $\mathbf{p}$ *weakly dominates* $\mathbf{q}$, denoted as $\mathbf{p} \preceq \mathbf{q}$, if $p_i \leq q_i$ for every $i \in \{1, \ldots, N\}$. We denote by $\mathbf{p} \npreceq \mathbf{q}$ if it does not hold that $\mathbf{p} \preceq \mathbf{q}$ and say that $\mathbf{p}$ and $\mathbf{q}$ are *mutually undominated* if both $\mathbf{p} \npreceq \mathbf{q}$ and $\mathbf{q} \npreceq \mathbf{p}$. Finally, we say that path π *weakly dominates* path π', denoted as $\pi \preceq \pi'$ if $\mathbf{c}(\pi) \preceq \mathbf{c}(\pi')$.

A search instance is defined as a tuple $P = (S, E, \mathbf{c}, s_{\text{start}}, s_{\text{goal}})$, where $(S, E, \mathbf{c})$ is a search graph and $s_{\text{start}}, s_{\text{goal}} \in S$ are the *start* and *goal* states, respectively. Given a search instance P, a *Pareto-optimal solution set* to s_{goal} from s_{start}, denoted as $sols(s_{\text{goal}})$, contains every path π from s_{start} to s_{goal} with the property that, for every other path π' from s_{start} to s_{goal}, $\pi' \npreceq \pi$ holds; that is, $sols(s_{\text{goal}})$ contains all non-dominated paths from s_{start} to s_{goal}. In this paper, we aim to find any maximal subset of the Pareto-optimal solution set such that their costs are mutually undominated. We refer to this subset as a *cost-unique Pareto-optimal solution set*.

Checking for weak dominance is a key operation in MOSP algorithms. In these algorithms, this operation takes a very specific form which we call *set dominance check*.

Problem 1 (Set Dominance Check (SDC))**.** *Given a set* $\mathbf{X}$ *of* N*-dimensional tuples (with* $N \geq 2$*) and a new* N*-dimensional tuple* $\mathbf{p}$*, the SDC problem calls to verify whether there exists a vector* $\mathbf{q} \in \mathbf{X}$ *such that* $\mathbf{q} \preceq \mathbf{p}$*.*

3 Related Work

In early work, Guerriero and Musmanno (2001) state that "*parallel computing [. . .] represents the main goal for future developments [in multi-objective search algorithms]*". However, there is little work on parallel MOSP algorithms.

Sanders and Mandow (2013) introduce a parallel variant of one of the early algorithms for the bi-objective setting (Martins 1984). Their work focuses on theoretical asymptotic running time and heap operations as the primary source of parallelism. Though promising, their work lacks experimental validation. Erb, Kobitzsch, and Sanders (2014) subsequently present a parallel bi-objective shortest-path algorithm utilizing weight-balanced B-trees with bulk updates. Their empirical evaluation demonstrates significant speedups, building upon the seemingly impractical work by Sanders and Mandow (2013).

Medrano and Church (2015) propose another parallel approach for computing the set of extreme solutions in the bi-objective setting, showing applicability using personal machines and shared memory supercomputers. Ahmadi et al. (2021) suggest a bi-objective bi-directional search algorithm where one search runs from the source and another from the target. Despite parallel execution, no substantial empirical improvement is reported. de las Casas et al. later adopt a similar approach for both bi-objective (de las Casas et al. 2021) and multi-objective (de las Casas, Sedeño-Noda, and Borndörfer 2021) algorithms.

In summary, the parallelization of multi-objective search remains a somewhat under-explored and challenging domain, despite being identified as a goal over twenty years ago. In this work we offer the first method to test for weak dominance via instruction-level parallelism. Our approach is simple-yet-effective and allows to obtain dramatic speedup using parallelism at the instruction level by processing multiple data elements with a single instruction.

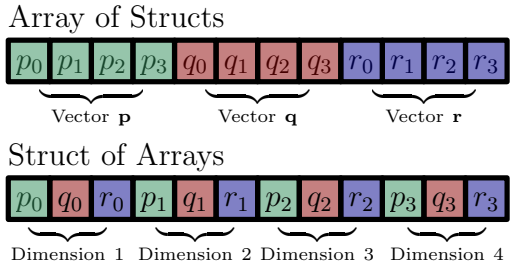

Figure 1: Array of Structs vs. Struct of Arrays. Adapted from (Thomason, Kingston, and Kavraki 2023).

4 Algorithmic Background

4.1 Vectorized Operations

Vectorized operations refer to performing operations on entire arrays or vectors of data at once, without explicit looping over individual elements. This approach is often implemented using the Single Instruction, Multiple Data (SIMD) paradigm. In SIMD, the same operation is applied to multiple data elements simultaneously, exploiting parallelism at the instruction level This is particularly useful in scenarios where the same operation needs to be performed on a large set of data elements, such as mathematical operations on arrays or pixels in image processing (Zhou and Ross 2002).

Programming languages and compilers often provide support for SIMD instructions, allowing developers to seamlessly write vectorized operations. Moreover, most modern CPUs support such instructions allowing SIMD-based code to run on any modern computer.[1]

Importantly, while vectorized operations can provide significant performance benefits, it requires careful consideration of data alignment which often corresponds to changing data representation from AoS to SoA (Intel 2019): AoS (Array of Structures) and SoA (Structure of Arrays) refer to different memory layout strategies for organizing data in computer programs, especially in the context of parallel computing and vectorization. The key difference lies in how the elements of a collection (such as an array or a structure) are stored in memory. In AoS, elements of a structure are stored contiguously in memory, and an array is used to store multiple instances of the structure. In contrast, in SoA, each component of a structure is stored in a separate array. Multiple arrays are then used to store the corresponding components of different structures. For a visualization, see Fig. 1.

4.2 Linear Time MOA* (LTMOA*)

In this section we describe Linear Time MOA* (LTMOA*)[2] with pseudo-code provided in Alg. 1. Similar

[1] Our implementation is based on the `AVX512` architecture which allows to process 512 bits simultaneously (Intel 2023).

[2] We describe a simplified version of LTMOA* where we do not perform dimensionality reduction (DR). DR reduces the number of dimensions considered when performing SDC by one. Our approach for SDC using instruction-level parallelism are immediately

Algorithm 1: LTMOA*

Input: A search problem $(S, E, \mathbf{c}, s_{\text{start}}, s_{\text{goal}})$ and a consistent heuristic function $\mathbf{h}$
Output: A cost-unique Pareto-optimal solution set

1: $sols \leftarrow \emptyset$
2: **for all** $s \in S$ **do**
3: $\quad \mathbf{G}_{\text{cl}}(s) \leftarrow \emptyset$
4: $n \leftarrow$ new node with vertex s_{start}
5: $\mathbf{g}(n) \leftarrow \mathbf{0}$; $\quad \mathbf{f}(n) \leftarrow \mathbf{h}(s_{\text{start}})$;
6: Initialize OPEN and add n to it
7: **while** OPEN $\neq \emptyset$ **do**
8: $\quad n \leftarrow$ OPEN.pop() $\quad$ // lex. min f-value
9: $\quad$ **if** IsDominated$(\mathbf{g}(n), \mathbf{G}_{\text{cl}}(s(n)))$ or IsDominated$(\mathbf{f}(n), \mathbf{G}_{\text{cl}}(s_{\text{goal}}))$ **then**
10: $\quad\quad$ **continue**
11: $\quad$ RemoveDominated$(\mathbf{g}(n), \mathbf{G}_{\text{cl}}(s(n)))$
12: $\quad$ Add $\mathbf{g}(n)$ to $\mathbf{G}_{\text{cl}}(s(n))$
13: $\quad$ **if** $s(n) = s_{\text{goal}}$ **then**
14: $\quad\quad$ Add n to $sols$
15: $\quad\quad$ **continue**
16: $\quad$ **for all** $s' \in$ Succ$(s(n))$ **do**
17: $\quad\quad$ $n' \leftarrow$ new node with vertex s' and parent n
18: $\quad\quad$ $\mathbf{g}(n') \leftarrow \mathbf{g}(n) + \mathbf{c}(s(n), s(n'))$
19: $\quad\quad$ $\mathbf{f}(n') \leftarrow \mathbf{g}(n') + \mathbf{h}(s(n'))$
20: $\quad\quad$ **if** IsDominated$(\mathbf{g}(n'), \mathbf{G}_{\text{cl}}(s(n)))$ or IsDominated$(\mathbf{f}(n'), \mathbf{G}_{\text{cl}}(s_{\text{goal}}))$ **then**
21: $\quad\quad\quad$ **continue**
22: $\quad\quad$ Add n' to OPEN
23: **return** $sols$

to other MOS algorithms, LTMOA* runs an A*-like best-first search using a priority list called OPEN containing the frontier of the search tree (i.e., the generated but not-yet-expanded nodes) and a set of solutions $sols$. Here, a node n is associated with a state $s(n) \in S$, a $\mathbf{g}$-value $\mathbf{g}(x)$ and a parent node. We also define $\mathbf{f}(x) = \mathbf{g}(x) + \mathbf{h}(s(x))$ as the $\mathbf{f}$-value of x. Conceptually, n corresponds to a path from s_{start} to $s(n)$ with cost $\mathbf{g}(n)$. The path can be constructed by backtracking along the parent nodes.

The search maintains $\mathbf{G}_{\text{cl}}(s)$, a set of non-dominated $\mathbf{g}$-values for each state s.[3] Every iteration, LTMOA* extracts the node n with the smallest lexicographic order and tests if its g-value is weakly dominated by any element in $\mathbf{G}_{\text{cl}}(s)$ or if its f-value is weakly dominated by any element in $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$ (Line 9). If one of this conditions holds, n is discarded. If n is not discarded, LTMOA* removes all $\mathbf{g}$-values dominated by $\mathbf{g}(n)$ from $\mathbf{G}_{\text{cl}}(s(n))$ (Line 11) and inserts $\mathbf{g}(n)$ to $\mathbf{G}_{\text{cl}}(s(n))$ (Line 12). It also checks if $s(n)$ is a goal state in which case it adds n to the set of solutions.

LTMOA* then expands n by generating a new node n' for every successor of $s(n)$. Similar to when extracting a node from OPEN, the algorithms tests if n''s g-value is weakly dominated by any element in $\mathbf{G}_{\text{cl}}(s)$ or if f-value is weakly dominated by any element in $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$ and if so, discards n' (Lines 20). If not, n' is added to OPEN.

Critical to our work is how LTMOA* (and other MOSP algorithms) implements SDC operations (Prob. 1 an Lines 9 and 20). This is done by iterating over all elements $\mathbf{q} \in \mathbf{X}$ and for each one testing if it dominates $\mathbf{p}$. See Alg. 2.

Algorithm 2: IsDominated

Input: A vector $\mathbf{p}$ and a set of vectors $\mathbf{X}$
Output: $true$ or $false$

1: **for all** $\mathbf{q} \in \mathbf{X}$ **do**
2: $\quad$ **if** $\mathbf{p} \preceq \mathbf{q}$ **then**
3: $\quad\quad$ **return** $true$
4: **return** $false$

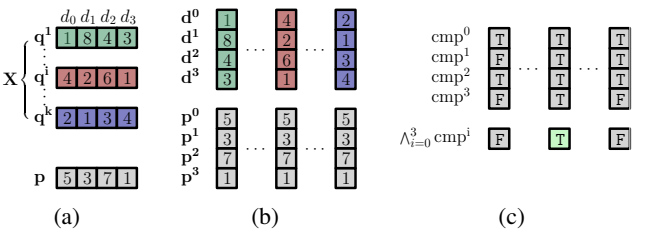

Figure 2: Example of performing vectorized SDC. (a) Classical data representation of a vector $\mathbf{p}$ and a set $\mathbf{X}$ of k vectors for $N = 4$ dimensions. (b) Vectorized data of $\mathbf{X}$ using an SOA representation. (c) Vectorized SDC.

5 MOSP using Vectorized SDC

5.1 Vectorized Set Dominance Checks

In this section we describe an alternative approach to implementing SDC using vectorized operations. Before describing our approach we denote n_{vec} to be the number of elements that can be stored in each vectorized operation[4] and $|\mathbf{X}| = k$ to be the number of elements in $\mathbf{X}$. To simplify the exposition, we assume that $n_{\text{vec}} = k$ but in practice we typically have that $n_{\text{vec}} \ll k$ and the following approach is repeated $\lceil n_{\text{vec}}/k \rceil$ times.

Data representation Key to our approach is changing the way our data ($\mathbf{X}$ and $\mathbf{p}$ in Prob. 1) is represented. We store $\mathbf{X}$ as an SoA and not as an AoS (Fig. 1). Specifically, $\mathbf{X}$ will now contain N elements $\mathbf{d^0}, \ldots \mathbf{d^N}$, such that $d_i^j := q_j^i$. Additionally (and with a slight abuse of notation), $\mathbf{p}$ is now stored as N elements $\mathbf{p^0}, \ldots \mathbf{p^N}$ such that $\mathbf{p^j}$ contains n_{vec} copies of the j'th element of $\mathbf{p}$. Namely, $\forall j \; p_i^j := p_j$.

To demonstrate this change in data representation, consider Fig. 2a and 2b. In classical AoS representation $\mathbf{q^i} = (4, 2, 6, 1)$, the i'th tuple in $\mathbf{X}$ (red cells) is stored sequentially. In our SoA representation, recall that $\mathbf{q_j^i}$, the j'th element in $\mathbf{q^i}$ is stored in d_i^j. For example, here, $d_i^2 = 6$. Similarly, in classical data representation, our query $\mathbf{p} = (5, 3, 7, 1)$ (grey cells), is stored sequentially. In our setting, recall that $\mathbf{p}$ is now stored as N elements $\mathbf{p^0}, \ldots \mathbf{p^N}$. For example, here, $p^1 = (3, \ldots, 3)$.

applicable to the setting where DR is used.

[3] Here 'cl' stands for "closed list".

[4] In typical systems, $n_{\text{vec}} \in \{8, 16, 32\}$.

	t_{mean}	t_{max}	t_{min}	t_{median}
	NY 3 Objectives			
	$\lvert sols\rvert_{\text{(mean, max, min)}}$ = (7,893, 49,870, 403)			
LTMOA*-C	136.7	556.3	1.6	18.2
LTMOA*-V	**38.0**	**164.9**	**0.7**	**5.5**
	NY 4 Objectives			
	$\lvert sols\rvert_{\text{(mean, max, min)}}$ = (22,650, 75,653, 360)			
LTMOA*-C	1,386.0	14,768.9	0.2	119.4
LTMOA*-V	**268.0**	**2,710.3**	**0.2**	**25.1**

(a)

	t_{mean}	t_{max}	t_{min}	t_{median}
	NY 3 Objectives			
	$\lvert sols\rvert_{\text{(mean, max, min)}}$ = (7,893, 49,870, 403)			
LTMOA*-C+DR	113.9	465.4	1.2	14.5
LTMOA*-V+DR	**29.4**	**129.4**	**0.6**	**3.8**
	NY 4 Objectives			
	$\lvert sols\rvert_{\text{(mean, max, min)}}$ = (22,650, 75,653, 360)			
LTMOA*-C+DR	1,098.9	11,867.0	0.2	104.0
LTMOA*-V+DR	**191.5**	**1,998.0**	**0.18**	**19.0**

(b)

Table 1: Running time in seconds without (a) and with (b) dimensionality reduction.

Algorithm 3: `IsDominated (Vectorized)`

Input: Vectors $\mathbf{p_0} \dots \mathbf{p_{N-1}}$ and vectors $\mathbf{d_0} \dots \mathbf{d_{N-1}}$
Output: $true$ or $false$

1: **for all** $\mathbf{j} \in \{1, \dots, N-1\}$ **do**
2: $\quad \text{cmp}_\text{j} \leftarrow \mathbf{d^j} \overset{?}{\leq} \mathbf{p^j}$ `vector of T's and F's`
3: **if** $\left(\bigwedge_{j=0}^{N} \text{cmp}^\text{j}\right)$ contains `T` **then**
4: $\quad$ **return** $true$
5: **return** $false$

Set Dominance Checks To perform SDC, we test if each element in $\mathbf{d_j}$ is smaller or equal than $\mathbf{p_j}$ to obtain cmp_j, a vector of `T`'s and `F`'s corresponding to a positive and negative answer, respectively. This is done in a single operation (Line 2 of Alg. 3). Subsequently, a logical and is performed over all these results (Lines 3-4 of Alg. 3) and if there exists a `T` in the result, the answers is positive.

Returning to our example where $\mathbf{q^i} = (4, 2, 6, 1)$ and $\mathbf{p} = (5, 3, 7, 1)$ we have that $\mathbf{p^i} \preceq \mathbf{q}$. Thus, the i'th value in cmp_j is set to `T` for every j. Consequently, the i'th value in $\left(\bigwedge_{j=0}^{N} \text{cmp}^\text{j}\right)$ contains `T` (light green cell in Fig. 2c).

5.2 Incorporating Vectorized SDC in LTMOA*

LTMOA* uses SPC when a node is popped from OPEN and when it is created. This corresponds to Lines 9 and 20 in Alg. 1, respectively. Importantly, if a node n is not weakly dominated after being popped from OPEN, LTMOA* removes all nodes in $\mathbf{G}_{\text{cl}}(s(n))$ that are dominated by $g(n)$. As we store $\mathbf{G}_{\text{cl}}(s(n))$ as an SoA, this may be a time-consuming operation. To this end, we remove this step (Line 11 in Alg. 1). This means that when using vectorized SDC, $\mathbf{G}_{\text{cl}}(s(n))$ may contain dominated nodes. This does not affect the correctness but may cause the algorithm to compute unnecessary dominance checks. As we demonstrate empirically, even this simplified approach allows to obtain speed ups when compared to existing implementations. We leave the problem of removing elements from an SoA data representation of $\mathbf{G}_{\text{cl}}(s(n))$ for future work.

6 Empirical Evaluation

Our code is based on the original C implementation of LTMOA* provided by the authors and is publicly available.[5] We use the same three and four objectives as reported in their empirical evaluations and compare vectorized and classical implementations of SPC[6] referring to these as LTMOA*-V and LTMOA*-C, respectively. We also incorporated dimensionality reduction (DR) and refer to these versions as LTMOA*-V+DR and LTMOA*-C+DR, respectively. We used a 2.80GHz Intel(R) Core(TM) i7-1165G7 CPU Linux laptop with 64GB of RAM. We use the NY map of the 9th DIMACS Implementation Challenge: Shortest Path[7].

In our implementation, we use `int` to represent each cost value which allows storing 16 tuples in each vector. While the fastest implementations of LTMOA* pre-allocates a fixed memory block for each $\mathbf{G}_{\text{cl}}$ set, this is not straightforward to implement when using SIMD instructions and requires care. Thus, we only pre-allocate a fixed memory block for $\mathbf{G}_{\text{cl}}(s_{\text{goal}})$ and leave more efficient implementations for future work.

We report in Table 1 the average, maximal minimal and median running times of results taken over 25 different random instances of the NY data set. We also report in Fig. 3 the runtime of individual instances for three and four objectives with DR. As we can see, using vectorized operations allows to dramatically speed up the running time of LTMOA*. In particular for the harder instances, the speedup in runtime is between $6\times$ and $8\times$.

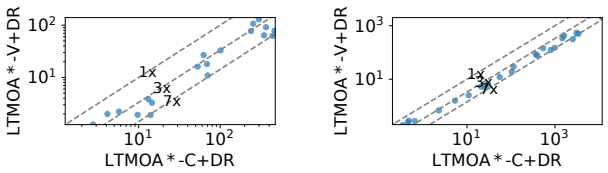

Figure 3: Runtime (in seconds) on individual instances comparing LTMOA*-V+DR and LTMOA*-C+DR on three (left) and four (right) objectives. Diagonal dashed lines denote different speed-ups.

[5] https://github.com/carlos-hu70/moavectorizated.

[6] We chose not to compare to alternative methods that employ paralelization for multi-objective search as they cannot be seamlessly integrated into state-of-the-art MOS algorithms and their code is not publicly available.

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

Acknowledgements

The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant no. 1817189, 1837779, 1935712, 2121028, 2112533, and 2321786. It has also been supported in part by the Israel Science Foundation (ISF) grant no. 909/23, the United States-Israel Binational Science Foundation (BSF) grants no. 2019703 and 2021643 and by the Israeli Ministry of Science & Technology grants No. 3-16079 and 3-17385. Finally, Carlos Hernández was supported by the National Center for Artificial Intelligence CENIA FB210017, Basal ANID, and the Centro Ciencia & Vida FB210008, Financiamiento Basal ANID

References

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

Bachmann, D.; Bökler, F.; Kopec, J.; Popp, K.; Schwarze, B.; and Weichert, F. 2018. Multi-Objective Optimisation Based Planning of Power-Line Grid Expansions. *ISPRS I. J. of Geo-Inf.*, 7(7): 258.

Breugem, T.; Dollevoet, T.; and van den Heuvel, W. 2017. Analysis of FPTASes for the Multi-Objective Shortest Path Problem. *Comput. Oper. Res.*, 78: 44–58.

Bronfman, A.; Marianov, V.; Paredes-Belmar, G.; and Lüer-Villagra, A. 2015. The Maximin HAZMAT Routing Problem. *Eur. J. Oper. Res.*, 241(1): 15–27.

de las Casas, P. M.; Kraus, L.; Sedeño-Noda, A.; and Borndörfer, R. 2021. Targeted Multiobjective Dijkstra Algorithm. *CoRR*, abs/2110.10978.

de las Casas, P. M.; Sedeño-Noda, A.; and Borndörfer, R. 2021. An Improved Multiobjective Shortest Path Algorithm. *Comput. Oper. Res.*, 135: 105424.

Ehrgott, M. 2005. *Multicriteria Optimization (2nd ed.)*. Springer.

Erb, S.; Kobitzsch, M.; and Sanders, P. 2014. Parallel bi-objective shortest paths using weight-balanced b-trees with bulk updates. In *SEA*, 111–122.

Fu, M.; Kuntz, A.; Salzman, O.; and Alterovitz, R. 2019. Toward Asymptotically-Optimal Inspection Planning via Efficient Near-Optimal Graph Search. In *RSS*.

Fu, M.; Salzman, O.; and Alterovitz, R. 2021. Computationally-Efficient Roadmap-Based Inspection Planning via Incremental Lazy Search. In *ICRA*, 7449–7456.

Guerriero, F.; and Musmanno, R. 2001. Label correcting methods to solve multicriteria shortest path problems. *J. Opt. Theory App.*, 111(3): 589–613.

Hansen, P. 1980. Bicriterion path problems. In *Multiple criteria decision making theory and application*, 109–127. Springer.

Hernández, C.; Yeoh, W.; Baier, J. A.; Zhang, H.; Suazo, L.; Koenig, S.; and Salzman, O. 2023. Simple and efficient bi-objective search algorithms via fast dominance checks. *Artif. Intell.*, 314: 103807.

Hernandez, C. U.; Salzman, O.; Baier, J. A.; Yeo, W.; Felner, A.; Kumar, S.; and Koenig, S. 2023. Multi-objective search via Lazy and Efficient Dominance checks. In *IJCAI*, 7223–7230.

Intel. 2019. Memory Layout Transformations. https://www.intel.com/content/www/us/en/developer/articles/technical/memory-layout-transformations.html.

Intel. 2023. Intel® 64 and IA-32 Architectures Software Developer's Manual. Technical report.

Martins, E. Q. V. 1984. On a special class of bicriterion path problems. *Eur. J. Oper. Res.*, 17(1): 85–94.

Medrano, F. A.; and Church, R. L. 2015. A parallel computing framework for finding the supported solutions to a biobjective network optimization problem. *J. Multi-Criteria Decis. Anal.*, 22(5-6): 244–259.

Pulido, F. J.; Mandow, L.; and De la Cruz, J. L. P. 2015. Dimensionality Reduction in Multiobjective Shortest Path Search. *Comput. Oper. Res.*, 64: 60–70.

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.

Sanders, P.; and Mandow, L. 2013. Parallel label-setting multi-objective shortest path search. In *Int. Symp. on Parallel and Distributed Processing*, 215–224.

Serafini, P. 1987. Some considerations about computational complexity for multi objective combinatorial problems. In *Recent advances and historical development of vector optimization*, 222–232. Springer.

Thomason, W.; Kingston, Z.; and Kavraki, L. E. 2023. Motions in Microseconds via Vectorized Sampling-Based Planning. *arXiv:2309.14545*.

Ulungu, E.; and Teghem, J. 1991. Multi-objective shortest path problem: A survey. In *Workshop on Multicriteria Decision Making: Methods–Algorithms–Applications*, 176–188.

Zhou, J.; and Ross, K. A. 2002. Implementing database operations using SIMD instructions. In *ACM SIGMOD international conference on Management of data*, 145–156.