Deeper Treatment of the Bi-objective Search Framework

Shawn Skyler[1], Dor Atzmon[2], Ariel Felner[1], Oren Salzman[3], Carlos Hernández Ulloa[4], Sven Koenig[5, 6]

[1]Ben-Gurion University of the Negev
[2]Bar-Ilan University
[3]Technion – Israel Institute of Technology
[4]Universidad San Sebastian
[5]University of California, Irvine
[6]Örebro University

shawn@post.bgu.ac.il, dor.atzmon@biu.ac.il, felner@bgu.ac.il, osalzman@cs.technion.ac.il, carlos.hernandez@uss.cl, sven.koenig@uci.edu

Abstract

In Bi-Objective Search (BOS), the task is to compute the Pareto-optimal frontier of paths in a graph with two cost values per edge. Recent work introduced a general BOS framework that classifies search nodes and studies how ordering functions affect expansion order. In this paper, we continue this line of research. We further refine the classes of nodes and show that many nodes that were added to the open list and are classified as never-expand nodes still need to be extracted and further examined. Additionally, we introduce a method that enables constant-time dominance checks for the MIN and MAX ordering functions. This allows a practical usage of these ordering functions, as we demonstrate in our experimental section.

1 Introduction

A* (Hart, Nilsson, and Raphael 1968) is an optimal and efficient algorithm (Dechter and Pearl 1985) for solving shortest-path *Single-Objective Search* (SOS) problems, optimizing a single cost metric. *Multi-Objective Search* (MOS) generalizes SOS to the case where multiple, often conflicting objectives exist. In MOS, we seek a set of Pareto-optimal solutions—paths that are not dominated across all objectives. Applications include pathfinding with competing criteria such as time and fuel, balancing cost and environmental impact, and robotic inspection planning (Bachmann et al. 2018; Fu et al. 2019; Fu, Salzman, and Alterovitz 2021).

MOS significantly increases complexity due to vector-valued costs and frequent dominance checks. Numerous MOS algorithms have been proposed (Mandow and Pérez-de-la-Cruz 2010; Clímaco and Pascoal 2012; Current and Marsh 1993; Skriver 2000; Tarapata 2007; Ulungu and Teghem 1991; Ren et al. 2025), and recent surveys identify ongoing challenges in pruning, optimality, and search efficiency (Salzman et al. 2023, 2026). A key special case, *Bi-Objective Search* (BOS), with exactly two objectives, has received particular attention (Ahmadi et al. 2021; Skyler et al. 2022; Hernández et al. 2023). While our work focuses on BOS, the results generalize to broader MOS contexts.

A recent theoretical framework for MOS and BOS was presented by Skyler et al. (2024) (denoted SKY24). It encapsulates a family of search algorithms that select nodes from the set of mutually undominated nodes in the open list (OPEN), using an *ordering function* (OF) to guide expansion. A common OF is lexicographic ordering (LEX) (Hernández et al. 2023), and SKY24 proposed two new functions, MIN and MAX. They showed that all OFs expand the same set of nodes but in different orders, affecting the timing of solution discovery. SKY24 also classified nodes into *must-expand*, *maybe-expand*, and *never-expand*.

A central operation in MOS/BOS algorithms is the *dominance check* (DC), used to prune dominated nodes. Since each node may be compared against many others, DC is computationally expensive. A key advantage of LEX in BOS is its support for constant-time DC (Hernández et al. 2023). In contrast, no such efficient mechanism was known for MIN and MAX, severely limiting their practical utility.

This paper provides two main contributions. First, we refine the classification of *never-expand nodes* (NENs). In SOS, *surplus nodes* that are never expanded remain in OPEN harmlessly (Felner et al. 2010). We show that in BOS, NENs must be extracted and checked for dominance, implying that search termination requires an empty OPEN. We further distinguish between NENs that are inserted and extracted from OPEN (and thus incur a DC), and those that are never inserted. Second, we introduce constant-time DC techniques for MIN and MAX, extending the efficiency previously available only to LEX. This allows all three OFs to be compared on equal footing as we show experimentally.

2 Definitions and Background

A *Bi-Objective Search* (BOS) graph G is a tuple $\langle S, E, \mathbf{c}\rangle$, where S, E are the states and edges and $\mathbf{c} : E \rightarrow \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ is a *cost function* in the form of a pair of two non-negative costs. A *path* π from v_1 to v_m is a sequence of states $v_1, v_2, \ldots, v_m$ such that $(v_i, v_{i+1}) \in E$. **Boldface** fonts represent pairs in the form $\mathbf{v} = (v_1, v_2)$. Given two pairs $\mathbf{u}$ and $\mathbf{v}$, we say that pair $\mathbf{u}$ *dominates* pair $\mathbf{v}$, denoted as $\mathbf{u} \prec \mathbf{v}$, if either (1) $u_1 \leq v_1$ and $u_2 < v_2$; or (2) $u_1 < v_1$ and $u_2 \leq v_2$. Otherwise, pair $\mathbf{u}$ *undominates* pair $\mathbf{v}$, denoted as $\mathbf{u} \not\preceq \mathbf{v}$.

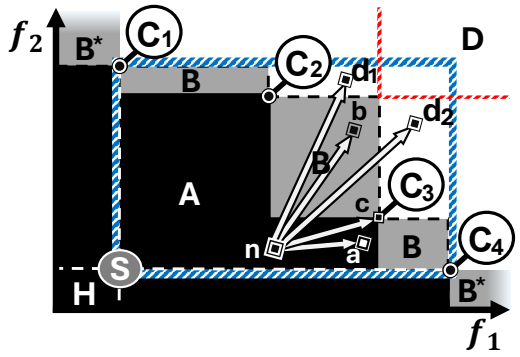


Figure 1: Different zones in a BOS instance. Every point in the figure is a two-dimensional cost from $start$ to $goal$.

For example, $(1,2) \prec (1,3)$ (as $2 < 3$). Finally, two pairs $\mathbf{u}$ and $\mathbf{v}$ are *mutually undominated* if both $\mathbf{u} \not\preceq \mathbf{v}$ and $\mathbf{v} \not\preceq \mathbf{u}$. For example, $(4,8)$ and $(6,6)$ are *mutually undominated*. A *search instance* is a tuple $I = \langle S, E, \mathbf{c}, start, goal, \mathbf{h} \rangle$, where $\langle S, E, \mathbf{c} \rangle$ is a BOS graph, $start$ and $goal$ are states $\in S$, and $\mathbf{h}$ is a heuristic function. A *solution* is a path from $start$ to $goal$. A solution is *Pareto-Optimal* if its cost is undominated by the cost of any other solution. A *Pareto-Optimal solution set*, denoted by POS, is the set of Pareto-Optimal solutions. Let $\mathcal{POC} = \{\mathbf{c}(\pi) \mid \pi \in POS\}$ denote the set of path costs in POS. A *cost-unique Pareto-Optimal solution set*, or *Pareto-Optimal Frontier*, denoted POF, is a subset of POS with no duplicate costs: there is one path for each value in $\mathcal{POC}$. In the special case of SOS, POS includes all optimal-cost paths, while POF includes only one. Fig. 1 illustrates cost pairs for a BOS instance. The C points denote $\mathcal{POC}$, with no solution costs in Zones $A \cup B$, and solutions in C are undominated by those in Zone D. In this paper, we focus on computing the POF.

A *single-valued* heuristic assigns to each state a pair $\mathbf{h} = \{h_1, h_2\}$ where each value h_i estimates the cost along dimension i. $\mathbf{h}$ is *admissible* if h_i is a lower bound of all the costs of dimension i in any solution. Similarly, $\mathbf{h}$ is *consistent* iff (*i*) $\mathbf{h}(goal) = \mathbf{0}$ and (*ii*) $\mathbf{h}(s) \preceq \mathbf{c}(s,t) + \mathbf{h}(t)$ for all $(s,t) \in E$. The strongest consistent single-valued heuristic commonly used is the *(single) point heuristic* (Goldin and Salzman 2021; Hernández et al. 2023; Pulido, Mandow, and Pérez-de-la Cruz 2015; Skyler et al. 2022). In the point heuristic, $h_i(n)$ is cost of the cost-minimal path from n to $goal$ using the i-th objective only. It is represented by the S node in the bottom left of Fig. 1.

2.1 Bi-Objective Search A* (BOS-A*)

BOS-A* is a general best-first framework coined by SKY24 for BOS algorithms. BOS-A* encapsulates most existing algorithms mentioned above for BOS. In BOS-A*, OPEN nodes are associated with $\mathbf{f}$-values. Thus, the *front* of OPEN includes all nodes with *undominated* ($\not\preceq$) $\mathbf{f}$-pairs. *front* can include multiple $\mathbf{f}$-pairs (e.g., $(1,2), (2,1)$), each can be used for choosing nodes for expansion. A node is chosen from *front* for expansion according to an *ordering function* (OF) (Skyler et al. 2022). OF defines a total order between

Algorithm 1: BOS-A*

```
1  BOS-A* (G = ⟨S, E, c⟩, start, goal, h, OF)
2      Init n_root, OPEN with OF; POF ← ∅
3      initialize_values()
4      s(n_root)=start; g(n_root) = 0; h(n_root) = h(start)
5      OPEN.insert(n_root)
6      while OPEN ≠ ∅ do
7          n ← OPEN.pop() // according to OF
8          if dominance_check(n) then
9              continue
10         update_values (n)
11         if s(n) = goal then
12             POF ← POF ∪ {n}
13             continue
14         for s' ∈ S s.t. (s(n), s') ∈ E do
15             Init n'; s(n') ← s'; h(n') = h(s');
                 g(n') = g(n) + c((s, s'))
16             if dominance_check(n') then
17                 continue
18             OPEN.insert(n')
19     return POF
```

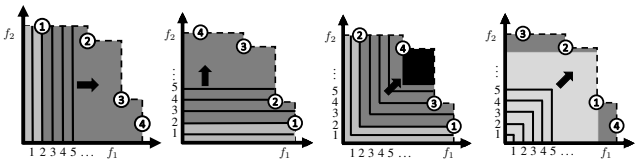
the different $\mathbf{f}$-pairs. LEX is an OF commonly used as it has various time and memory advantages (Hernández et al. 2023). We discuss and compare LEX and other OFs below.

BOS-A* (Alg. 1) receives a BOS instance as well as OF. Each node n contains a state $s(n)$, and a $\mathbf{g}$-value pair $\mathbf{g}(n)$. $\mathbf{f}(n)$ is calculated as $\mathbf{f}(n) = \mathbf{g}(n) + \mathbf{h}(s(n))$. In each iteration, BOS-A* extracts a node n with the *"best"* $\mathbf{f}$-value from the *front* of OPEN, as determined by OF. BOS-A* then performs the common expansion cycle and either adds a node to POF if $s(n) = goal$ or expands it. When expanding a node n, it generates a child node for each successor s' of $s(n)$. When OPEN is empty, it halts and returns the POF.

2.2 Dominance Checks (DCs)

BOS-A* performs *dominance checks* (DCs) for node n to determine whether n has the potential to be included in POF. Otherwise, n is pruned. DC is performed in two locations: after a node was chosen for expansion (Line 8) and after a node was generated but before it is inserted to OPEN (Line 16). Two types of DCs are performed on n: **(1:) *Path Dominance Check*** (PDC). In PDC, $\mathbf{g}(n)$ is compared against the $\mathbf{g}$-value of a list of other nodes m that share the same state, i.e., $s(m) = s(n)$. We only keep the undominated occurrences of the $\mathbf{g}$-values. PDC is equivalent to *Duplicate Detection* in SOS. PDC is optional — one may choose not to implement it and risk a significant increase in the size of OPEN and thus the CPU time. **(2:) *Solution Dominance Check*** (SDC). In SDC, $\mathbf{f}(n)$ is compared against the list of values in $\mathcal{POC}$. If $\mathbf{f}(n)$ is dominated by one of these values, it is pruned. SDC is mandatory. Without it, the algorithm can run forever and continue to explore nodes. SDC does not have an equivalent operation in SOS.

Both DC types iterate over potentially large lists — previous instances for PDC or solution costs (POC) for SDC —



(a) LEX1 (b) LEX2 (c) MIN (d) MAX

Figure 2: Progress of different OFs.

to check node dominance. Thus, in general, DCs run in time linear to the size of these lists, which can be very large and impose substantial CPU overhead. Occurrences of BOS-A* (i.e., many of the algorithms in the literature, such as the ones referenced above) differ in whether, when, and how DCs are implemented, as well as the data structures used for maintaining OPEN. In some BOS cases, however, DC can be performed in constant time (Hernández et al. 2023), resulting in significant running time reduction.

2.3 Ordering Functions

In the BOS-A* framework, any node belonging to the *front* of OPEN (i.e., all mutually undominated nodes) can be selected for expansion. To decide which node to expand first, BOS-A* employs an *ordering function* (OF). This section reviews several OFs, highlighting how each OF influences the search progression as illustrated in Fig. 2. The circled numbers indicate the order of solutions found. The Light Gray region shows the portion of the cost space explored until the *first* solution is discovered, the Darker Gray region covers expansions performed after the first solution but before all Pareto-optimal solutions are found, and the Black region (relevant only for MIN) indicates expansions that occur *after* the algorithm has already identified the entire POF. This final step in MIN ensures that no additional Pareto-optimal solutions remain. We next cover different OFs.

Lexicographical Ordering (LEX). LEX prioritizes a single objective and uses the other objective to break ties (Hernández et al. 2023). LEX has two variants: LEX1, which compares f_1-values first, and LEX2, which compares f_2-values first. Assume three $\mathbf{f}$-values: $(1,5)$, $(4,2)$, $(3,3)$. LEX1 prioritizes $(1,5)$, $(3,3)$, $(4,2)$ while LEX2 prioritizes $(4,2)$, $(3,3)$, $(1,5)$. Fig. 2a shows how LEX1 expands solutions with smaller f_1 values first, proceeding from the leftmost side of the cost space toward higher f_1 values. Similarly, Fig. 2b depicts LEX2, which starts at the bottom of the cost space (lowest f_2) and moves upward. A key advantage of LEX is that its DCs can be performed in constant time by maintaining the smallest relevant cost of the secondary dimension (Hernández et al. 2023). Notably, LEX explores the POF sequentially and solutions near the opposite end of the frontier are found late in the search.

Maximum and Minimum Ordering (MAX, MIN). Skyler et al. (2022) introduced two additional OFs: MAX and MIN. In MAX, each $\mathbf{f}(n)$ is first arranged in decreasing order (i.e., the larger among the two costs of $f(n)$ is placed first) and then compared lexicographically (LEX1). Assume $\mathbf{f}$-values: $(1,5)$, $(4,2)$, $(3,3)$, MAX prioritizes $(3,3)$, $(4,2)$, $(1,5)$ (according to the minimal max-value). As shown in Fig. 2d, MAX initially focuses on "middle-range" solutions and expands outward toward both extremes. This may require exploring a broader region of the cost space, thereby incurring a larger initial overhead. Conversely, MIN sorts each node's cost pair in *increasing order* before comparing them lexicographically. Thus, MIN prioritizes $(1,5)$, $(4,2)$, $(3,3)$. Fig. 2c illustrates that MIN interleaves the behaviors of LEX1 and LEX2, starting from both extremes of the POF and converging toward the center. By prioritizing whichever dimension is smaller for each node, MIN often recognizes the entire POF faster than other OFs. However, because MIN focuses first on extremes, more balanced solutions are discovered later in the search. The Black region in Fig. 2c reflects MIN's final check after all Pareto-optimal solutions have been found, ensuring that no additional undominated paths exist.[1]

Normalization. When objectives differ in scale (e.g., time in seconds vs. distance in miles), normalization is crucial for fair comparisons in MAX and MIN. Let $\min_i$ be the minimum cost of dimension i among edges emanating from the start state: $\min_i = \min_{(start,v)\in E} c_i(start, v)$, and let $\max_i = \max_{\mathbf{c}\in\mathcal{POC}} c_i$ be an upper bound on dimension i among Pareto-optimal solutions. Each cost x_i is linearly projected into $[0,1]$ via $\bar{x}_i = \frac{x_i - \min_i}{\max_i - \min_i}$. Anchoring the normalization to the minimal costs at the start state and maximal costs in POF ensures a total order of pairs of values.

2.4 Classes of Nodes in SOS

In SOS, nodes can be divided into three categories: **(1:)** *Must-expand nodes (MENs)*: nodes that must be expanded to guarantee completeness or optimality. These nodes have $f < C^*$. **(2:)** *Maybe-expand nodes (MBENs)*: nodes that may or may not be expanded, depending on how tie-breaking proceeds. These nodes have $f = C^*$. **(3:)** *Never-expand nodes (NENs)*: nodes that will never be expanded as they cannot lead to a better solution than the one that was found. These nodes have $f > C^*$. Importantly, some NENs are generated and added to OPEN but they will never be touched again. These were designated as *surplus nodes* (Felner et al. 2010). In SOS, once the algorithm finds an optimal solution and verifies that no better path can appear in OPEN, it can halt, even if OPEN still includes (surplus) nodes.

2.5 Classes of Nodes in BOS

In BOS, the task is to find the POF, not just a single solution as in SOS. Thus, SKY24 generalized the classification of nodes to four classes which correspond to four distinct zones: *A*, *B*, *C*, and *D* (Fig. 1), described next. Notably, in BOS, there are two classes of MENs (Zones A and B).

Zone A (Optimality MENs): The Black area in Fig. 1. It contains MENs that are essential for discovering and prov-

[1] BOBA* (Ahmadi et al. 2021) is a bidirectional search for BOS where the frontiers do not meet but the searches are executed all the way to the other end. BOBA* is not directly comparable with our work as it is not a member of BOS-A* because it is a bidirectional search. Future work can implement MIN and MAX on top of BOBA* and explore its must-expand theory discussed below.

ing the optimality of the POF. These correspond to MENs in SOS. A node n is an *optimality MEN* if it dominates one solution from any POF, i.e., $\exists \mathbf{p} \in \mathcal{POC} \mid \mathbf{f}(n) \prec \mathbf{p}$.

Zone B (Completeness MENs): The Gray area in Fig. 1. It contains additional MENs required to guarantee the completeness of the POF search. Completeness nodes n are mutually undominated by any solution in any POF. That is, a node n is a *completeness MEN* if $\forall \mathbf{p} \in \mathcal{POC}$ it holds that $\mathbf{f}(n) \not\preceq \mathbf{p}$ and $\mathbf{p} \not\preceq \mathbf{f}(n)$. Unlike Zone A, Zone B does not have a direct analogy in SOS. Nodes in Zone B have to be expanded to ensure that the POF is complete.

Zone C (MBENs): The magnified dots in Fig. 1. Node n is a *maybe-expand node* if $\exists \mathbf{p} \in \mathcal{POC}$ such that $\mathbf{f}(n) = \mathbf{p}$. All C points in Fig. 1 are MBENs and analogous to nodes with $f(n) = C^*$ in SOS. These nodes, whose cost pairs appear in the POF, may not necessarily correspond to goal states. Their expansion depends on the tie-breaking policy used.

Zone D (NENs): The White area in Fig. 1. It contains nodes whose costs are dominated by at least one solution in POF. That is, a node n is a NEN if $\exists p \in \mathcal{POC}$ such that $p \prec \mathbf{f}(n)$. These correspond to NEN in the SOS context.

3 Deeper Treatment of NENs

SKY24 showed that all OFs must expand the same set of nodes (MEN+MBEN), albeit in a different order. Thus, OFs may reach solutions in a different order and at a different pace, e.g., MIN finds the entire POF earlier in the search than LEX. Nevertheless, SKY24 overlooked some attributes of NENs (Zone D). While all nodes in Zone D are never expanded, they are not *surplus* in the sense that those nodes that were inserted to OPEN must be cleared from OPEN before termination. This section examines this issue.

3.1 Life Cycle of a Node

In best-first search algorithms (in SOS, BOS, and MOS), it is common to say that when nodes are *expanded*, their successors are *generated*. However, there are additional steps that occur during the life cycle of a node n, as follows.

Step 1 (generation): An operator is applied on the expanded parent node p and a new node n is being *generated* (by copying p and performing the operator on it).

Step 2 (check): Node n is *checked* and might be pruned by the optional *Duplicate Detection* in SOS, by the PDC in BOS/MOS, or by the mandatory SDC in BOS/MOS. If the node is dominated, it is pruned. This step can be done eagerly after a node is generated (Step 2) and/or lazily after the node is extracted from OPEN (Step 5).

Step 3 (insertion): If node n was not pruned, it is being *inserted* to and maintained in OPEN for further treatment.

Step 4 (extraction): Node n is chosen to be *extracted* from OPEN based on its $\mathbf{f}$-value (and on the OF and tie-breaking rule that are used in case of BOS/MOS).

Step 5 (second check): Similar to Step 2, the extracted node n is again *checked* and might be pruned.

Step 6 (goal test): Node n undergoes a *goal test*. If n is a goal then, in SOS, the algorithm halts while, in BOS/MOS, n is added to POF.

Step 7 (expansion): If the extracted node n is not a goal, it is *expanded*. That is, we apply all relevant operators in order to generate each of its successors (Step 1).

As mentioned above, in SOS, NENs (with $f > C^*$) might be inserted to OPEN but they will never be touched again, i.e., they will never be chosen for extraction. The reason is that they can be potentially chosen for extraction/expansion only if their f-value is the best in OPEN. But this will not happen because the search halts when a goal node with $f = C^*$ is (or all goal nodes are) extracted. Thus, NENs are designated as *surplus* as they stay intact in OPEN. Notably, NENs are being inserted to OPEN because, at the time of their generation, the value of C^* is yet unknown.

By contrast, BOS involves identifying all Pareto-optimal solution costs ($\mathcal{POC}$) rather than a single solution cost. The algorithm halts only after verifying that no additional Pareto-optimal solution exists. Thus, we cannot stop after a given solution is found because other solutions may exist, necessitating the complete exhaustion of OPEN to assure that no unexplored Pareto-optimal paths remain.

In BOS, every generated node must undergo at least one SDC (Steps 2 and 5). Consider a given node n in Zone D. Each such node is dominated by at least one solution node C_i. Therefore, SDC against C_i prunes node n. But, at the time of generation, if the goal C_i is not yet recognized, performing SDC will not help and n will be inserted to OPEN. However, unlike SOS, those nodes from Zone D in OPEN must later be extracted (and pruned) as we explain next.

We must perform a SDC on all nodes in OPEN because they might belong to each of the four Zones A, B, C, D. An example is given in Fig. 1. Node n in Zone A is being expanded while a solution of cost C_2 was already found. Five successors of n are generated in the different zones and in the following order: d_1 is generated, but a DC realizes that it is dominated by C_2 and thus d_1 is pruned. Next, d_2 is generated. But, since C_3 was not yet found, d_2 is not dominated by any existing solution and, thus, d_2 is inserted to OPEN. Then, the three nodes a, b, and c are generated and not pruned by any existing solution. Thus, they are all inserted to OPEN. Both a and b will be extracted and expanded. c must be extracted with any OF before d_2 because it dominates it. Since c is a goal, the solution of C_3 will be added to POF. Next, d_2 is extracted and immediately pruned by the solution of C_3. This phenomenon (of nodes from Zone D being inserted into OPEN only to be pruned upon extraction) is observed for all OFs, including LEX.

3.2 Active Region Within Zone D

With the single-point heuristic, the *extreme solutions* (C_1, C_4 in Fig. 1) can be computed with a SOS algorithm in a preprocessing phase (Skyler et al. 2022). This partitions the search space into two according to the Blue frame in Fig. 1:

(1) Inactive Region (outside the frame (C_1, D, C_4, S)**).** Recall that S is the $\mathbf{f}$-value of $start$. Thus, no nodes to its left or below will ever be generated. If a node is generated above or right of the frame, it is pruned because it is clearly dominated by C_1 or C_4. A constant-time DC against C_1 and C_4 can yield this without requiring a more sophisticated DC.

Algorithm 2: LEX constant-time DC

```
1  initialize_values ()
2      for s ∈ S do
3          g_2^min(s) ← ∞
4  dominance_check (Node n)
5      if f_2(n) ≥ g_2^min(goal) then
6          return True // SDC
7      if g_2(n) ≥ g_2^min(s(n)) then
8          return True // PDC
9      return False // undominated
10 update_values (Node n)
11     g_2^min(s(n)) ← g_2(n)
```

Note that, the boundary regions B^* can be pruned this way.

(2) Active Region (inside the frame (C_1, D, C_4, S)). These nodes must undergo the regular DC and may be inserted to OPEN and later extracted during the search. If they are in Zones A and B, they will also be expanded.

As shown by SKY24, all OFs expand exactly the same set of nodes. Naturally, all of their neighbors are generated and, thus, all OFs generate the same set of nodes. However, OFs affect the *order* by which nodes are expanded and, thus, the order and rate by which solutions are discovered. This may be crucial in two ways. First, in an *anytime scenario*, where one needs many solutions as fast as possible. In addition, different OFs may treat generated nodes differently. Second, assume that a node n is generated in Zone D, which is dominated by a solution in C_i. Some OFs might already know about the solution in C_i and prune it upon generation without inserting it into OPEN. Other OFs might not yet know about C_i and insert node n into OPEN, and then extract it and prune it due to C_i later. Therefore, the number of *extracted* nodes varies with the choice of OF, and this might affect the running time as a node in Zone D that is generated but skips OPEN incurs less overhead than a node in Zone D that is inserted and extracted from OPEN.

Overall, understanding the interplay between expansions and extractions is critical for evaluating the performance of BOS algorithms. In scenarios that demand fast discovery of multiple solutions (e.g., anytime settings), an OF that minimizes unnecessary extractions might speed up the search.

4 Constant-Time Dominance Checks

DC is expensive because a node is usually matched against a possibly very large list. Hernández et al. (2023) presented the BOA* algorithm, a variant of BOS-A*, that performs DC for LEX in constant time rather than linear time. Below, we introduce constant-time DC for MIN and MAX, thus making them competitive to LEX. Note that, all constant-time DCs require a consistent heuristic.

4.1 LEX Constant-Time DCs

We describe the constant-time DC for LEX1 (Alg. 2), used within Alg. 1; LEX2 is symmetrically defined by swapping the objectives. A key observation (Hernández et al. 2023) is that under LEX1, g_1 for any state s must strictly increase across expansions, requiring g_2 to monotonically decrease to avoid domination. Based on that, constant-time DC has three parts: initialization, value updates, and dominance checks.

`initialize_values`: To perform a constant-time DC in LEX1 (Alg. 2), we maintain a g_2^{min} value for each state s. $g_2^{min}(s)$ stores the minimal g_2 value seen for s among the different nodes n that have s as their state. $g_2^{min}(s)$ is initialized to ∞ (Lines 1-3).

`dominance_check`: Let n be the currently explored node. SDC returns *True* if $f_2(n) \geq g_2^{min}(goal)$, i.e., the path through n cannot be developed into a path that is undominated by all of the already found goals (Lines 5-6). PDC returns *True* if $g_2(n) \geq g_2^{min}(s(n))$, i.e., a prior path was found that dominates the path of n (Lines 7-8).

`update_values`: If node n is not pruned, then $g_2^{min}(s(n))$ is updated with $g_2(n)$ (Line 11).

Example of PDC: Let $(1, 9)$, $(3, 4)$, $(5, 7)$, $(6, 6)$, $(7, 5)$, $(8, 2)$ be pairs of costs of nodes (paths) leading to the same state s. These costs are presented in the order by which they are explored by LEX1. For $(1, 9)$, `dominance_check` returns $False$ and g_2^{min} is set to 9. Then, for $(3, 4)$, again `dominance_check` returns $False$ and $g_2^{min} \leftarrow 4$. For $(5, 7)$, $(6, 6)$, and $(7, 5)$, it holds that $g_2(n) \geq g_2^{min}(s)$, they are pruned, and *True* is returned (Lines 7-8). Lastly, for $(8, 2)$, `dominance_check` returns $False$ and $g_2^{min} \leftarrow 2$. SDC is done in a similar manner, except that, for each node n, we now compare $f_2(n)$ with $g_2^{min}(goal)$.

4.2 Constant-Time DC for MIN and MAX

Since LEX1 progresses along dimension 1 (see Fig. 2a), it only requires maintaining a single value (g_2^{min}) to perform constant-time DC. Importantly, while nodes are expanded based on their $\mathbf{f}$-values, in LEX, it suffices to maintain and compare $\mathbf{g}$-values for performing DC. In fact, it is possible to maintain (f_2^{min}) and compare $\mathbf{f}$-values (instead of $\mathbf{g}$-values) for the DC of LEX (Alg. 2). But, it will have no impact because, in PDC, a state always has the same $\mathbf{h}$-value and, in SDC, $\mathbf{h}(goal) = (0, 0)$. Thus, the order by which nodes are visited and whether a node is pruned or not will not be affected. By contrast, MIN and MAX progress along both dimensions (see Figs. 2c and 2d). We classify nodes n into two classes: Class L_1 where $f_1(n) < f_2(n)$ and class L_2 where $f_1(n) > f_2(n)$ (nodes with $f_1(n) = f_2(n)$ can be in either class). Thus, here, we need to maintain two values, one for each class. However, here, we cannot maintain $\mathbf{g}$-values to prune nodes and at the same time use $\mathbf{f}$-values to prioritize nodes. The reason is that a node can be classified to L_1 based on $\mathbf{g}(n)$ but to L_2 based on $\mathbf{f}(n)$. For example, consider two nodes p and q with $\mathbf{h}(s(p)) = \mathbf{h}(s(q)) = (1, 100)$, $\mathbf{g}(p) = (1, 2)$, and $\mathbf{g}(q) = (2, 1)$. Both nodes are classified to L_1 by their $\mathbf{f}$-values, but q belongs to L_2 based on $\mathbf{g}$-values due to the heuristic asymmetry. To address this, we propose for MIN and MAX storing f_1^{min} and f_2^{min} (and not g_i^{min}, as in LEX) and then prune nodes based on their $\mathbf{f}$-values. This ensures a consistent classification of nodes since both pruning and prioritization are based on $\mathbf{f}$-values.

Algorithm 3: MIN constant-time DC

```
1  initialize_values ()
2      for s ∈ S do
3          f_1^min(s) ← ∞; f_2^min(s) ← ∞
4  dominance_check (Node n)
5      if f_1(n) ≥ f_1^min(goal) ∥ f_2(n) ≥ f_2^min(goal) then
6          return True // SDC
7      if f_1(n) ≥ f_1^min(s(n)) ∥ f_2(n) ≥ f_2^min(s(n)) then
8          return True // PDC
9      return False // undominated
10 update_values (Node n)
11     if f_1(n) ≤ f_2(n) then
12         f_2^min(s(n)) ← f_2(n)
13     if f_1(n) ≥ f_2(n) then
14         f_1^min(s(n)) ← f_1(n)
```

Algorithm 4: MAX constant-time DC

```
1  initialize_values ()
2      for s ∈ S do
3          f_1^min(s) ← ∞; f_2^min(s) ← ∞
4  dominance_check (Node n)
5      if f_1(n) ≥ f_1^min(goal) & f_2(n) ≥ f_2^min(goal) then
6          return True // SDC
7      if f_1(n) ≥ f_1^min(s(n)) & f_2(n) ≥ f_2^min(s(n)) then
8          return True // PDC
9      return False // undominated
10 update_values (Node n)
11     if f_1(n) < f_1^min(s(n)) then
12         f_1^min(s(n)) ← f_1(n)
13     if f_2(n) < f_2^min(s(n)) then
14         f_2^min(s(n)) ← f_2(n)
```

4.3 MIN Constant-Time DCs

Alg. 3 presents the DC functions in constant time for MIN, which should be used within the general Alg. 1.

initialize_values: MIN selects the node with smallest $\min(f_1, f_2)$. Hence, each state keeps two minima f_1^{min} and f_2^{min}, initialized to ∞ (Line 3).

dominance_check: SDC (and PDC) are in Lines 5-8. Given node n, $f_1(n)$ is compared with $f_1^{min}(goal)$ (and $f_1^{min}(s(n))$, resp.), and $f_2(n)$ is compared with $f_2^{min}(goal)$ (and $f_2^{min}(s(n))$, resp.). If one of these inequalities holds, then n is pruned. We prove the correctness of this rule next.

update_values: If the current node was not pruned by the expansion's DC, L_1 nodes ($f_1 < f_2$) update f_2^{min} (Lines 11-12); L_2 nodes update f_1^{min} (Lines 13-14).

Theorem 1. *Assume that node n is chosen for expansion. There exists another node m such that $s(m) = s(n)$ and $m \prec n$, iff the conditions in Line 7 prune n (for PDC).*

Proof. **Direction 1 $\Leftarrow$:** Assume w.l.o.g that n belongs to L_1 (i.e., $(f_1(n) \leq f_2(n))$. There are now two cases.

Case 1: Let $m_1 \in L_1$ be the node that triggered the value of $f_2^{min}(s(n))$ (recall that $s(n) = s(m_1)$). Assume that the right condition of Line 7 holds. This means that $f_2(m_1) \leq f_2(n)$. Now, since m_1 belongs to L_1 and since it was expanded before n, then $f_1(m_1) \leq f_1(n)$. Thus, $m_1 \prec n$.
Case 2: Let m_2 be the L_2 node that triggered the value of $f_1^{min}(s(n))$. Assume that the condition in the left side of Line 7 holds. This means that $f_1(m_2) \leq f_1(n)$.
Now observe that $f_2(n) \geq f_1(n) \quad (n \in L_1)$
$\geq f_1(m_2) \quad (Line\ 7)$
$\geq f_2(m_2) \quad (m_2 \in L_2).$
Therefore, $m_2 \prec n$.

Direction 2 $\Rightarrow$: If neither node m prunes n, no other node can, since their f_2 (f_1) is the minimal value.
Identical proofs are for the $n \in L_2$ case and for SDC. □

Example of PDC: Let $(1, 9)$, $(8, 2)$, $(3, 4)$, $(5, 7)$, $(7, 5)$, $(6, 6)$ be a sequence of **f**-costs seen for the same state s, in the order explored by MIN. For $(1, 9)$, which belongs to L_1, dominance_check returns $False$ and f_2^{min} is set to 9. Then, for $(8, 2)$, which belongs to L_2, again dominance_check returns $False$ and $f_1^{min} \leftarrow 8$. For $(3, 4)$, dominance_check also returns $False$ and $f_2^{min} \leftarrow 4$. For $(5, 7)$, $(7, 5)$, $(6, 6)$, it holds that $f_2(n) \geq f_2^{min}$, they are pruned, and *True* is returned (Lines 7-8).

We experimentally found that such cases are relatively rare and exploiting this idea had a negligible effect on the running time.

4.4 MAX Constant-Time DCs

The main difference between the constant-time DC of MAX (Alg. 4) and MIN (Alg. 3) is that the conditions for the PDC and SDC have *and* operation (Lines 5 and 7 in Alg. 4) as MAX prunes nodes with values *larger than or equal to* the corresponding f^{min} values. Examples for MAX and its proof are similar to those of MIN and are omitted.

5 Experimental Results

We experimented on the Bay, New York, and Colorado roadmaps from the DIMACS challenge (DIMACS 2006) using the common travel time and distance as objectives. We implemented BOS-A* with BOA* and evaluated the different OFs. BOA* prunes nodes lazily when they are extracted from OPEN. Alternatively, a node can prune nodes when it is inserted into OPEN, resulting in a smaller OPEN.

Table 1 compares total runtimes for MIN and MAX using linear versus our new constant-time DC, averaged over 100 instances per map. Similar to earlier findings for LEX (Hernández et al. 2023), constant-time DC yields roughly a two-order magnitude runtime improvement, making MIN and MAX practically competitive with LEX.

Table 2 summarizes average operation counts on 575 BAY instances. Confirming SKY24, all OFs performed exactly 123,541 expansions. However, the OFs differed slightly (within $\approx 6\%$) in other metrics, such as extractions, DCs, percolations (shifts in OPEN), and runtime. Variations oc-

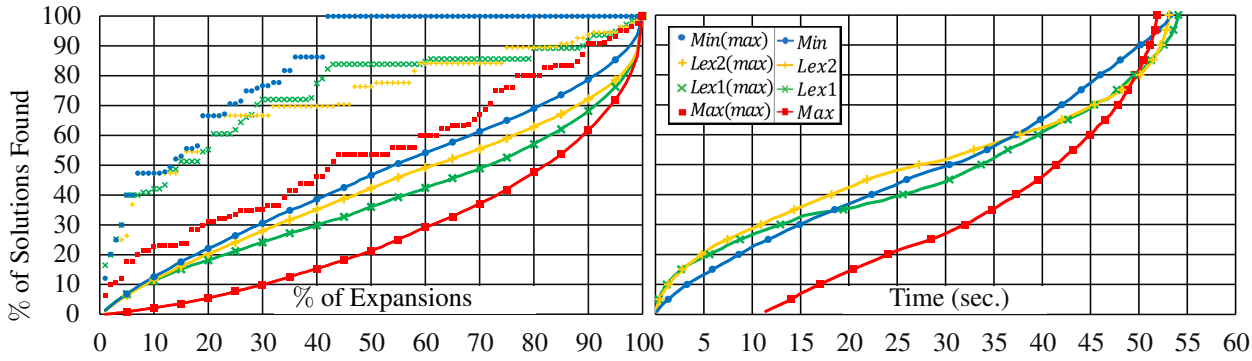


Figure 3: Left: Solutions found as a function of percentage of expansions. Right: Solutions found as a function of time (sec).

Time	Bay		New York		Colorado	
	MIN	MAX	MIN	MAX	MIN	MAX
Linear	61	59	5,189	4,567	7,068	6,338
Constant	≤1	≤1	32	31	42	42

Table 1: Linear vs. constant-time DC runtimes (secs).

OF	Extract	Percolations	DCs	Compare	Time
LEX1	156,554	2,679,386	493,157	954,897	54.06
LEX2	146,828	2,232,956	483,430	926,095	53.11
MAX	155,081	2,729,033	491,683	1,834,796	51.86
MIN	147,784	2,588,078	484,386	1,684,661	53.35

Table 2: Operations on BAY (123K expansions for all OFs).

cur because nodes may be pruned earlier or later depending on the chosen OF. Notably, *comparisons* for MIN and MAX are roughly double those of LEX due to maintaining both f_1^{min} and f_2^{min} rather than a single scalar. Nevertheless, total runtimes for all OFs remained comparable, as constant-time DC mitigates these overheads.

Fig. 3(left) presents another view of this experiment. It shows the percentage of solutions found from the POF (y-axis) as a function of the ratio of the expansions (x-axis), measured in percentiles, averaged over all our cases. There is a curve for each of the OFs. Being close to the top left corner is desirable because this indicates that many solutions are found faster during the search. On average, MIN was the best in finding earlier parts of the POF, then LEX2 and LEX1, while MAX was the slowest. Additionally, the dots at the top of the figure present the maximum value (among the different runs) that the OF achieved for each percentile. The maximum values (dots) strongly correlate with the average results. In fact, there was an instance in which MIN found the entire POF after only 43% of the expansions. The remaining 57% expansions were to verify that no other solutions exist. To summarize, while LEX reaches the first solution the fastest (one of the extreme solutions), other functions (MIN) are able to find many solutions earlier in the search and reveal the entire POF earlier, with fewer expansions.

Only counting nodes expanded hides the CPU time per node for the different OFs. Thus, before having constant-time DC also for MIN and MAX, these OFs could only be compared theoretically (comparing nodes) as done in Fig. 3(left). Practically, MIN and MAX with linear-time DC could not compete with the constant-time DC of LEX (see Table 1). Fig. 3(right) shows the same experiment, but here, the x-axis is the time elapsed. Indeed, since all algorithms can now perform constant-time DC, the same trends as Fig. 3(left) are shown, but there is a slight shift in runtime (compared to the number of expansions) of $\approx 6\%$.

6 Practical Comparison of the OFs

Given similar total runtimes and our new constant-time DC implementation for MIN and MAX, all OFs are now on equal footing with LEX. So, choosing among them depends primarily on application-specific tradeoffs as we discuss next.

LEX: LEX is simple, quickly finding an extreme solution first. However, solutions at the opposite extreme are delayed since LEX proceeds systematically across the frontier. Additionally, differences between LEX1 and LEX2 can impact DC counts and runtime, depending on objective scaling.

MAX: MAX finds balanced (middle) solutions early, beneficial in anytime scenarios seeking immediate intermediate solutions. However, MAX delays extreme solutions, and requires more expansions initially.

MIN: MIN finds both extreme solutions very fast but intermediate balanced solutions appear later in the search. MIN generally provides rapid complete frontier discovery.

MAX and MIN necessitate objective normalization but offer stable performance independent of objective ordering.

7 Conclusions

We further developed the theory of NEN in BOS and showed that nodes inserted to OPEN must later be extracted before halting. Additionally, we introduced a constant-time DC for MIN and MAX, making them comparable and competitive to LEX. Experimental results on road networks show that MIN finds solutions earlier than other OFs and that, using constant-time DC, all OFs have similar runtimes for computing the entire Pareto-Optimal Frontier.

Possible directions for future work include (1) defining the general set of OFs for which a constant-time DC exists; and (2) extending these techniques to higher-dimensional MOS, which may further reduce runtime.

Acknowledgements

This research was supported by the United States-Israel Binational Science Foundation (BSF) grants no. 2021643. The work of Ariel Felner was supported by the Israel Science Foundation (ISF) grant #909/23. The work of Dor Atzmon was supported by the Israel Science Foundation (ISF) grant #1511/25 and by Israel's Ministry of Innovation, Science and Technology (MOST) grant #6908 (Czech-Israeli cooperative scientific research). Oren Salzman was supported in part by the Technion Autonomous Systems Program (TASP), and by the United States-Israel Binational Science Foundation (BSF) grant no. 2019703. Carlos Hernández was supported by the National Center for Artificial Intelligence CENIA FB210017, Basal ANID. The research at the University of California, Irvine was supported by the National Science Foundation (NSF) under grant numbers 2544613, 2434916, 2321786, 2112533, as well as gifts from Amazon Robotics and the Donald Bren Foundation.

References

Ahmadi, S.; Tack, G.; Harabor, D.; and Kilby, P. 2021. Bi-Objective Search with Bi-Directional A^*. In Mutzel, P.; Pagh, R.; and Herman, G., eds., *29th Annual European Symposium on Algorithms*, volume 204 of *LIPIcs*, 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 International Journal of Geo-Information*, 7(7): 258.

Clímaco, J.; and Pascoal, M. 2012. Multicriteria Path and Tree Problems: Discussion on Exact Algorithms and Applications. *ITOR*, 19(1-2): 63–98.

Current, J.; and Marsh, M. 1993. Multiobjective Transportation Network Design and Routing Problems: Taxonomy and Annotation. *EJOR*, 65(1): 4–19.

Dechter, R.; and Pearl, J. 1985. Generalized Best-First Search Strategies and the Optimality of A^*. *J. ACM*, 32(3): 505–536.

DIMACS. 2006. The 9th DIMACS Implementation Challenge: Shortest Path.

Felner, A.; Goldenberg, M.; Sharon, G.; Stern, R.; Beja, T.; Sturtevant, N.; Schaeffer, J.; and Holte, R. 2010. Partial-Expansion A* with Selective Node Generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1): 471–477.

Fu, M.; Kuntz, A.; Salzman, O.; and Alterovitz, R. 2019. Toward Asymptotically-Optimal Inspection Planning Via Efficient Near-Optimal Graph Search. In *Robotics: Science and Systems XV*.

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

Goldin, B.; and Salzman, O. 2021. Approximate Bi-Criteria Search by Efficient Representation of Subsets of the Pareto-Optimal Frontier. In *ICAPS*, 149–158.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

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.

Mandow, L.; and Pérez-de-la-Cruz, J. 2010. Multiobjective A^* search with consistent heuristics. *J. ACM*, 57(5): 27:1–27:25.

Pulido, F. J.; Mandow, L.; and Pérez-de-la Cruz, J.-L. 2015. Dimensionality Reduction in Multiobjective Shortest Path Search. *CAR*, 64: 60–70.

Ren, Z.; Hernández, C.; Likhachev, M.; Felner, A.; Koenig, S.; Salzman, O.; Rathinam, S.; and Choset, H. 2025. EMOA*: A framework for search-based multi-objective path planning. *Artificial Intelligence*, 339: 104260.

Salzman, O.; Felner, A.; Hernández, C.; and Koenig, S. 2026. Multi-Objective Search: Algorithms, Applications, and Emerging Directions. In *AAAI-2026*.

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-23*, 6759–6768. ijcai.org.

Skriver, A. J. 2000. A Classification of Bicriterion Shortest Path (BSP) Algorithms. *Asia Pacific J. of Oper. Res.*, 17(2): 199–212.

Skyler, S.; Atzmon, D.; Felner, A.; Salzman, O.; Zhang, H.; Koenig, S.; Yeoh, W.; and Hernández, C. 2022. Bounded-Cost Bi-Objective Heuristic Search. In *SoCS22*, 239–243.

Skyler, S.; Shperberg, S.; Atzmon, D.; Felner, A.; Salzman, O.; Chan, S.-H.; Zhang, H.; Koenig, S.; Yeoh, W.; and Hernandez Ulloa, C. 2024. Theoretical Study on Multi-objective Heuristic Search. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 7021–7028.

Tarapata, Z. M. 2007. Selected Multicriteria Shortest Path Problems: An Analysis of Complexity, Models and Adaptation of Standard Algorithms. *IJAM*, 17(2).

Ulungu, E.; and Teghem, J. 1991. Multi-Objective Shortest Path Problem: A Survey. In *Workshop on Multicriteria Decision Making*, 176–188.