Technical Companion to

Generalizing Multi-Objective Search via Objective-Aggregation Functions

A Self-Contained Guide for CS Undergraduates

Companion to the paper by Hadar Peer, Eyal Weiss,
Ron Alterovitz, and Oren Salzman

Abstract

This document explains every equation, algorithm, and technical concept in the paper *Generalizing Multi-Objective Search via Objective-Aggregation Functions* in detail, assuming only undergraduate-level knowledge of graph search (Dijkstra's algorithm, A*). No prior knowledge of multi-objective search, Pareto optimality, or aggregation functions is required. Each equation is accompanied by (1) a plain-English description of what it computes, (2) a symbol-by-symbol breakdown, (3) an intuitive explanation of why the computation makes sense, and (4) a concrete numerical example where helpful.

Contents

1 Background: Multi-Objective Optimization and Search

Before diving into the paper's contributions, we need to build up several prerequisite concepts. If you know single-objective shortest-path algorithms like Dijkstra or A*, you are in good shape — this section extends those ideas to settings with *multiple* objectives.

1.1 What Is Multi-Objective Optimization?

In standard (single-objective) optimization, we have one cost to minimize. For example, in shortest-path search, we minimize path length. There is always a clear "best" solution: the one with the lowest cost.

In **multi-objective optimization**, we want to minimize *several* costs simultaneously. For example, a robot planning a path might want to:

1. Minimize **path length** (get there quickly), and
2. Minimize **collision risk** (stay safe).

The trouble is that these objectives often *conflict*: a shorter path might cut through dangerous areas, while a safer path might be much longer. There is no single "best" solution — instead, there is a *set* of solutions that represent different trade-offs.

Key Idea

In multi-objective optimization, we do not seek a single optimal solution. Instead, we seek the set of all "non-dominated" solutions — those for which no other solution is strictly better on *every* objective. This set is called the **Pareto-optimal frontier** (POF).

1.2 Vectors and Dominance

Because we have multiple objectives, the "cost" of a path is no longer a single number — it is a **vector**.

Definition 1.1 (Cost vector)**.** For a problem with d objectives, the cost of a path π is a vector $\mathbf{c}(\pi) = (c_1, c_2, \ldots, c_d) \in \mathbb{R}^d_{\geq 0}$, where each c_i measures the path's performance on objective i.

Numerical Example

Suppose we have $d = 2$ objectives: path length and collision risk. Three paths have costs:

Path	Length (c_1)	Risk (c_2)
π_1	5	0.8
π_2	8	0.2
π_3	10	0.1

Path π_1 is short but risky; π_3 is safe but long; π_2 is in between.

We need a way to compare these cost vectors.

Definition 1.2 (Dominance $\preceq$)**.** We say that vector $\mathbf{p}$ **dominates** vector $\mathbf{q}$, written $\mathbf{p} \preceq \mathbf{q}$, if $\mathbf{p}$ is at least as good as $\mathbf{q}$ on *every* objective:

$$\mathbf{p} \preceq \mathbf{q} \quad \Longleftrightarrow \quad \forall\, i,\ p_i \leq q_i.$$

Symbol breakdown:

- p_i: the i-th component of vector $\mathbf{p}$.

- q_i: the i-th component of vector $\mathbf{q}$.
- $\forall\, i$: "for all objectives $i = 1, 2, \ldots, d$."
- $p_i \leq q_i$: $\mathbf{p}$ is at least as good (no worse) as $\mathbf{q}$ on objective i.

Intuition: If path A is both shorter *and* safer than path B, then A dominates B. There is no reason to ever choose B. But if A is shorter but riskier, then neither dominates the other — they represent genuinely different trade-offs.

Numerical Example

Using the paths above:

- Does π_1 dominate π_2? Compare $(5, 0.8)$ vs. $(8, 0.2)$: $5 \leq 8$ (yes), but $0.8 \leq 0.2$? No. So π_1 does **not** dominate π_2.
- Does π_2 dominate π_3? Compare $(8, 0.2)$ vs. $(10, 0.1)$: $8 \leq 10$ (yes), but $0.2 \leq 0.1$? No. So π_2 does **not** dominate π_3.
- Consider a fourth path $\pi_4 = (9, 0.5)$. Does π_2 dominate π_4? Compare $(8, 0.2)$ vs. $(9, 0.5)$: $8 \leq 9$ (yes) and $0.2 \leq 0.5$ (yes). So $\pi_2 \preceq \pi_4$ — yes, π_2 dominates π_4. Path π_4 is worse on *both* objectives, so it should be discarded.

1.3 Lexicographic Ordering

When we need to totally order cost vectors (e.g., to decide which node to expand next in a priority queue), dominance alone is not enough — many pairs are incomparable. The paper uses **lexicographic ordering**.

Definition 1.3 (Lexicographic ordering $\prec_{\text{lex}}$)**.** We say $\mathbf{p}$ is **lexicographically smaller** than $\mathbf{q}$, written $\mathbf{p} \prec_{\text{lex}} \mathbf{q}$, if at the first index k where $\mathbf{p}$ and $\mathbf{q}$ differ, we have $p_k < q_k$. More precisely: $\mathbf{p} \prec_{\text{lex}} \mathbf{q}$ if $p_k < q_k$ for the first index k such that $p_k \neq q_k$ (or if $\forall i, p_i \leq q_i$, i.e., $\mathbf{p}$ dominates $\mathbf{q}$).

Intuition: This is exactly how words are ordered in a dictionary. "apple" comes before "banana" because 'a' $<$ 'b' at the first position. Similarly, $(2, 7)$ comes before $(3, 1)$ because at the first component, $2 < 3$.

Numerical Example

- $(2, 5) \prec_{\text{lex}} (3, 1)$: first components differ, $2 < 3$.
- $(3, 1) \prec_{\text{lex}} (3, 4)$: first components equal, second components differ, $1 < 4$.
- $(1, 2, 5) \prec_{\text{lex}} (1, 2, 7)$: first two equal, third: $5 < 7$.

1.4 Approximate Dominance

Computing the exact Pareto-optimal frontier can be extremely expensive (the number of solutions can grow exponentially with the number of objectives). In practice, we often settle for an *approximation*.

Definition 1.4 (Approximate dominance $\preceq_\varepsilon$)**.** Let $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_d)$ be a vector with $\varepsilon_i \geq 0$. We say $\mathbf{p}$ **ε-dominates** $\mathbf{q}$, written $\mathbf{p} \preceq_\varepsilon \mathbf{q}$, if:

$$\forall\, i, \quad p_i \leq (1 + \varepsilon_i) \cdot q_i.$$

Intuition: Approximate dominance relaxes the comparison. A solution $\mathbf{p}$ "approximately dominates" $\mathbf{q}$ if $\mathbf{p}$ is within a factor of $(1 + \varepsilon_i)$ of $\mathbf{q}$ on each objective — even if $\mathbf{p}$ is slightly worse on some objectives, we consider it "good enough."

An **ε-approximate POF** Π^*_ε is a set of solutions such that every path in the true POF Π^* is ε-dominated by some path in Π^*_ε. This gives us a compact set that "covers" the entire true frontier.

Numerical Example

With $\varepsilon = (0.1, 0.1)$: does $\mathbf{p} = (5.4, 0.21)$ approximately dominate $\mathbf{q} = (5, 0.2)$?

- Objective 1: $5.4 \leq 1.1 \times 5 = 5.5$? Yes.
- Objective 2: $0.21 \leq 1.1 \times 0.2 = 0.22$? Yes.

So $\mathbf{p} \preceq_\varepsilon \mathbf{q}$: even though $\mathbf{p}$ is slightly worse on both objectives, it is within the 10% tolerance and approximately dominates $\mathbf{q}$. This means that if $\mathbf{q}$ is in the true POF, we can "represent" it by $\mathbf{p}$ in the approximate POF.

1.5 The Pareto-Optimal Frontier (POF)

Definition 1.5 (Pareto-Optimal Frontier)**.** Given a MOS graph $G = (V, E, \mathbf{c})$ with start vertex v_s and goal vertex v_g, a path π from v_s to v_g is called a **solution**. The **Pareto-optimal frontier** Π^* is the set of all solutions such that for every $\pi \in \Pi^*$, there does not exist another solution π' with $\pi' \preceq \pi$ (i.e., π' dominates π).

In plain English: The POF is the set of solutions that are not dominated by any other solution. These are the "best" trade-offs: you cannot improve one objective without worsening another.

Numerical Example

Consider the four paths from earlier:

Path	Length	Risk
π_1	5	0.8
π_2	8	0.2
π_3	10	0.1
π_4	9	0.5

π_4 is dominated by π_2 (since $8 \leq 9$ and $0.2 \leq 0.5$). No other path is dominated by another. Therefore: $\Pi^* = \{\pi_1, \pi_2, \pi_3\}$.
Visually, if you plot Length vs. Risk, the POF forms a "staircase" along the lower-left boundary — any point not on the staircase is dominated by some point on it.

1.6 Review: A* Search (Single-Objective)

Since the multi-objective algorithms in this paper are based on A*, let us quickly review how A* works in the familiar single-objective setting.

A* maintains:

- $g(n)$: the cost of the best known path from the start to node n.
- $h(n)$: a **heuristic** estimate of the cost from n to the goal.
- $f(n) = g(n) + h(n)$: the estimated total cost of the best path through n.
- OPEN: a priority queue of nodes to explore, ordered by f.
- CLOSED: the set of already-expanded nodes.

A* repeatedly extracts the node with the smallest f from OPEN, expands it (generating its neighbors), and continues until the goal is reached. If h is **admissible** (never overestimates the true cost), A* guarantees finding the optimal path.

From Single-Objective to Multi-Objective A*

The key difference in multi-objective search is:

1. g, f, and h become *vectors* instead of scalars.
2. Instead of keeping only the best path to each node, we may need to keep *multiple* paths (one for each non-dominated cost vector).
3. Instead of returning one solution, we return the entire Pareto-optimal frontier.

2 Problem Formulation: The MOS Graph

2.1 The MOS Graph

Definition 2.1 (MOS Graph)**.** A **multi-objective search (MOS) graph** is a tuple $G = (V, E, \mathbf{c})$ where:

- V is a finite set of **vertices** (nodes).
- $E \subseteq V \times V$ is a finite set of **edges**.
- $\mathbf{c} : E \to \mathbb{R}^d_{\geq 0}$ is a **cost function** that assigns each edge a d-dimensional vector of non-negative real costs.

Symbol breakdown:

- V: the set of all locations/states the robot (or agent) can be in.
- E: the set of valid transitions between locations.
- $\mathbf{c}$: assigns a cost vector to each transition. For example, if $d = 2$ and the objectives are length and risk, then $\mathbf{c}(e) = (\ell_e, r_e)$ gives the length and risk of traversing edge e.
- $\mathbb{R}^d_{\geq 0}$: the set of d-dimensional vectors with all non-negative entries.

Numerical Example

A small road network with 4 intersections and $d = 2$ objectives (distance in km, risk score):

Edge	Distance	Risk
(A, B)	3	0.1
(A, C)	5	0.0
(B, D)	4	0.3
(C, D)	2	0.05

Two paths from A to D:

- $A \to B \to D$: cost $= (3, 0.1) + (4, 0.3) = (7, 0.4)$.
- $A \to C \to D$: cost $= (5, 0.0) + (2, 0.05) = (7, 0.05)$.

The second path dominates the first (same distance, lower risk), so only $A \to C \to D$ is in the POF.

2.2 Path Cost

The cost of a path $\pi = v_1, v_2, \ldots, v_n$ is the sum of its edge costs:

$$\mathbf{c}(\pi) = \sum_{i=1}^{n-1} \mathbf{c}(v_i, v_{i+1}).$$

Important note: This is the *standard* definition where costs are additive. As we will see, many real-world applications violate this assumption — this is a key motivation for the paper's generalization.

Additivity Is Not Always True

In the standard MOS formulation, path cost is the sum of edge costs. But what if the path's "risk" is the probability of colliding with *any* obstacle? If the probability of avoiding obstacle 1 is 0.8 and avoiding obstacle 2 is 0.9, the probability of avoiding both is $0.8 \times 0.9 = 0.72$ (assuming independence). The risk is $1 - 0.72 = 0.28$. This is **not additive**: $0.2 + 0.1 = 0.3 \neq 0.28$. The paper's framework handles exactly such cases via aggregation functions.

3 The Core Contribution: Hidden Objectives and Aggregation

This section explains the paper's central idea: separating objectives into **hidden objectives** (tracked during search) and **solution objectives** (what we ultimately want to optimize), connected by aggregation functions.

3.1 Why Standard MOS Falls Short

Consider a robot navigating among q obstacles under uncertainty. We want to minimize both *path length* and *total collision risk* (the probability of hitting *any* obstacle).

Approach 1 (Naive standard MOS): Treat overall risk as one objective, path length as the other ($d = 2$). The problem: we cannot compute overall risk from a partial path because it depends on *all* obstacles, but we might not have encountered all of them yet.

Approach 2 (Brute-force standard MOS): Create one objective per obstacle ($d = q + 1$: one risk per obstacle plus path length). The problem: the POF size — and hence runtime — can grow *exponentially* in d. With 12 obstacles, the search can become intractable.

The Paper's Insight: Hidden vs. Solution Objectives

The paper's key insight is to separate two types of objectives:

1. **Hidden objectives** (m of them): fine-grained quantities tracked during the search. Example: per-obstacle collision risks, per-edge road type information.
2. **Solution objectives** (k of them, with $k < m$): coarser quantities that we actually want to optimize. Example: total collision risk, path length.

The solution objectives are defined as **aggregation functions** applied to the hidden objectives. By searching in the k-dimensional solution space (instead of the m-dimensional hidden space), the POF is much smaller, and search is much faster.

3.2 The Two Key Functions

The framework introduces two functions that generalize the standard MOS formulation.

3.2.1 Path-Cost Extension Function $\mathcal{F}_{\text{ext}}$

Definition 3.1 (Path-cost extension function)**.** A **path-cost extension function** $\mathcal{F}_{\text{ext}} : \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^m$ defines how the cost of a partial path is updated when a new edge is appended.

Given a path $\pi = v_1, \ldots, v_n$, let $\pi^i = v_1, \ldots, v_i$ be the partial path consisting of the first i vertices. The **extended cost** is defined recursively:

- $\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi^1) := \mathbf{0}$ (the cost of a single-vertex "path" is the zero vector).
- For $i > 1$: $\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi^i) := \mathcal{F}_{\text{ext}}(\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi^{i-1}),\ \mathbf{c}(v_{i-1}, v_i))$.

What it does in plain English: To compute the cost of a path, start with zero cost, then for each edge along the path, apply $\mathcal{F}_{\text{ext}}$ to combine the cost-so-far with the new edge's cost. This is a generalization of "just add the edge cost."

Symbol breakdown:

- $\mathbb{R}^m$: the space of **hidden objective** cost vectors (dimension m).
- $\mathbb{R}^d$: the space of edge costs (dimension d). Note: m and d can differ! The edge cost might encode raw information (e.g., per-obstacle risks plus length), while the hidden cost accumulates it in a different form.
- $\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi^{i-1})$: the accumulated hidden cost of the path so far (up to vertex v_{i-1}).
- $\mathbf{c}(v_{i-1}, v_i)$: the cost vector of the new edge being traversed.
- $\mathcal{F}_{\text{ext}}(\cdot, \cdot)$: the function that combines these to produce the new accumulated cost.

The trivial case: When $m = d$ and $\mathcal{F}_{\text{ext}}(\mathbf{g}, \mathbf{g}') = \mathbf{g} + \mathbf{g}'$ (component-wise addition), this reduces to the standard additive path cost.

Trivial (additive) extension

With $d = m = 2$ and additive $\mathcal{F}_{\text{ext}}$, consider a path $A \to B \to C$:

$$\begin{aligned}
\mathbf{c}_{\mathcal{F}_{\text{ext}}}(A) &= (0, 0) \\
\mathbf{c}_{\mathcal{F}_{\text{ext}}}(A \to B) &= \mathcal{F}_{\text{ext}}\big((0, 0),\ \mathbf{c}(A, B)\big) = (0, 0) + (3, 0.1) = (3, 0.1) \\
\mathbf{c}_{\mathcal{F}_{\text{ext}}}(A \to B \to C) &= \mathcal{F}_{\text{ext}}\big((3, 0.1),\ \mathbf{c}(B, C)\big) = (3, 0.1) + (4, 0.2) = (7, 0.3)
\end{aligned}$$

This is just summing edge costs, as expected.

3.2.2 Objective-Aggregation Function $\mathcal{F}_{\text{agg}}$

Definition 3.2 (Objective-aggregation function)**.** An **objective-aggregation function** $\mathcal{F}_{\text{agg}} : \mathbb{R}^m \to \mathbb{R}^k$ maps the m-dimensional hidden cost to a k-dimensional **solution cost**, where $k \leq m$.

What it does: Once a path reaches the goal, we apply $\mathcal{F}_{\text{agg}}$ to its hidden cost to obtain the final cost vector that we actually want to optimize. This is where the "aggregation" happens: multiple hidden objectives are combined into fewer solution objectives.

Definition 3.3 (Solution cost)**.** The **solution cost** of a complete path (solution) π is:

$$\mathbf{c}_{\mathcal{F}_{\text{agg}}}(\pi) := \mathcal{F}_{\text{agg}}(\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi)).$$

What it computes: First compute the hidden cost of the path ($\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi)$, using the extension function), then aggregate it into the solution cost ($\mathcal{F}_{\text{agg}}$).

The trivial case: When $\mathcal{F}_{\text{agg}}$ is the identity function (and $m = k$), the solution cost equals the hidden cost. This is the standard MOS setting.

Definition 3.4 (POF w.r.t. an aggregation function)**.** The **POF w.r.t.** $\mathcal{F}_{\text{agg}}$, denoted $\Pi^*_{\mathcal{F}_{\text{agg}}}$, is the set of all solutions π such that no other solution π' satisfies $\mathbf{c}_{\mathcal{F}_{\text{agg}}}(\pi') \prec \mathbf{c}_{\mathcal{F}_{\text{agg}}}(\pi)$.

In plain English: The POF is now defined in terms of the *solution* costs (after aggregation), not the hidden costs.

The POF Changes Depending on the Aggregation

If $\mathcal{F}_{\text{agg}}$ is the identity, we get the standard POF over all m objectives. With a non-trivial $\mathcal{F}_{\text{agg}}$ that maps to $k < m$ dimensions, the POF is defined in the k-dimensional solution space and is typically **much smaller**. This is the whole point: a smaller POF means faster search.

3.3 Monotonicity Requirement

The paper requires that $\mathcal{F}_{\text{ext}}$ be **monotonically non-decreasing**:

$$\mathbf{g} \preceq \mathbf{g}' \implies \mathcal{F}_{\text{ext}}(\mathbf{g}, \mathbf{c}) \preceq \mathcal{F}_{\text{ext}}(\mathbf{g}', \mathbf{c}) \quad \text{for all } \mathbf{c}.$$

In plain English: If cost-so-far $\mathbf{g}$ is dominated by $\mathbf{g}'$ (i.e., $\mathbf{g}$ is better on every component), then after extending by the same edge, $\mathbf{g}$ remains at least as good. Adding an edge cannot "flip" the ordering of two partial paths.

Why this matters: This property is essential for correctness of the search algorithm. It ensures that if one partial path dominates another *now*, it will continue to dominate after any future extension. Without this, pruning dominated paths would be unsafe — a currently-dominated path might become non-dominated after future edges.

Numerical Example

Additive extension trivially satisfies monotonicity: if $\mathbf{g} \preceq \mathbf{g}'$ (i.e., $g_i \leq g'_i$ for all i), then $\mathbf{g} + \mathbf{c} \preceq \mathbf{g}' + \mathbf{c}$ (since $g_i + c_i \leq g'_i + c_i$).

3.4 Concrete Examples of Non-Trivial Extension and Aggregation

The paper provides three examples from robotics. We explain each one in detail.

3.4.1 Example 1: Planning Under Obstacle Uncertainty (Equations 1–2)

Setting: A robot navigates among q obstacles. Each obstacle O_i has associated "shadows" — probabilistic regions encoding position uncertainty. The robot must minimize both *path length* and *total collision risk* (the probability of hitting at least one obstacle).

Hidden objectives: $m = q + 1$ hidden objectives:

- $o_1, \ldots, o_q$: the per-obstacle collision risk (one per obstacle).
- $o_{q+1} = c_m$: the accumulated path length.

Edge cost encoding: For each edge e, the cost vector $\mathbf{c}(e) = (c_1, c_2, \ldots, c_{m-1}, c_m)$ is d-dimensional (with $d = m$ for this application), where c_i for $i \in \{1, \ldots, m-1\}$ is the per-obstacle risk contribution of edge e for obstacle O_i, and c_m is the edge's path length.

Extension function (Equation 1):

$$\mathcal{F}_{\text{ext}}(\mathbf{c}, \mathbf{c}') = \big(\max(c_1, c'_1), \ldots, \max(c_{m-1}, c'_{m-1}),\ c_m + c'_m\big) \qquad \text{(Eq. 1)}$$

What it computes: For each obstacle, the extended cost is the *maximum* of the per-obstacle risk seen so far and the new edge's contribution (because risk corresponds to the closest approach, and taking the max tracks the worst-case intersection). For path length, costs simply add.

Symbol breakdown:

- c_i: the accumulated risk from obstacle i along the path so far.
- c'_i: the risk contribution from the new edge for obstacle i.
- $\max(c_i, c'_i)$: the worst (highest) risk seen so far for obstacle i.
- $c_m + c'_m$: path length accumulates additively.

Aggregation function (Equation 2):

$$\mathcal{F}_{\text{agg}}(\mathbf{c}) = \left(1 - \prod_{i=1}^{m-1}(1 - c_i),\ c_m\right) \tag{Eq. 2}$$

What it computes: The solution objectives are:

1. **Total collision risk**: the probability of colliding with *at least one* obstacle, computed from the per-obstacle risks assuming independence.
2. **Path length**: passed through unchanged.

Symbol breakdown:

- c_i: risk of hitting obstacle i (a value in $[0, 1]$).
- $1 - c_i$: probability of *not* hitting obstacle i.
- $\prod_{i=1}^{m-1}(1 - c_i)$: probability of not hitting *any* obstacle (assuming independence).
- $1 - \prod_{i=1}^{m-1}(1 - c_i)$: probability of hitting at least one obstacle = total risk.
- c_m: path length, unchanged.

This produces $k = 2$ solution objectives from $m = q + 1$ hidden objectives.

Numerical Example

Suppose $q = 3$ obstacles. A path has per-obstacle risks $c_1 = 0.3$, $c_2 = 0.2$, $c_3 = 0.1$ and path length $c_4 = 15$.
Aggregated cost:

$$\begin{aligned}\text{Total risk} &= 1 - (1 - 0.3)(1 - 0.2)(1 - 0.1) \\ &= 1 - (0.7)(0.8)(0.9) \\ &= 1 - 0.504 = 0.496.\end{aligned}$$

Solution cost: $(0.496, 15)$.
Compare with naive addition: $0.3 + 0.2 + 0.1 = 0.6$. The correct risk (0.496) is lower than the naive sum (0.6) because risk does not add linearly — hitting two obstacles is not twice as bad as hitting one, probability-wise.

Extension function step by step

Consider a path $A \to B \to C$ with $q = 2$ obstacles. Edge (A, B) has cost $\mathbf{c}(A, B) = (0.1, 0.0, 5)$: obstacle 1 risk is 0.1, obstacle 2 risk is 0.0, length is 5.
Edge (B, C) has cost $\mathbf{c}(B, C) = (0.3, 0.2, 3)$: obstacle 1 risk is 0.3, obstacle 2 risk is 0.2, length is 3.
Step 1: $\mathbf{c}_{\mathcal{F}_{\text{ext}}}(A) = (0, 0, 0)$.
Step 2: $\mathbf{c}_{\mathcal{F}_{\text{ext}}}(A \to B) = \mathcal{F}_{\text{ext}}\big((0, 0, 0),\ (0.1, 0.0, 5)\big) = \big(\max(0, 0.1),\ \max(0, 0.0),\ 0 + 5\big) = (0.1, 0.0, 5)$.

Step 3: $\mathbf{c}_{\mathcal{F}_{\text{ext}}}(A \to B \to C) = \mathcal{F}_{\text{ext}}\big((0.1, 0.0, 5),\ (0.3, 0.2, 3)\big) = \big(\max(0.1, 0.3),\ \max(0.0, 0.2),\ 5+3\big) = (0.3, 0.2, 8)$.
Final aggregation: $\mathcal{F}_{\text{agg}}(0.3, 0.2, 8) = \big(1-(1-0.3)(1-0.2),\ 8\big) = (1-0.56,\ 8) = (0.44,\ 8)$.
Solution cost: total risk is 0.44, path length is 8.

3.4.2 Example 2: Inspection Planning (Equation 3)

Setting: A robot must visit as many points of interest (POIs) as possible while minimizing path length. There are q POIs.

Hidden objectives: $m = q + 1$: one binary variable $o_i \in \{0, 1\}$ per POI (has it been seen?), plus path length.

Extension function: Same as Equation (1): the max operator for POI indicators (once seen, stays seen), and addition for path length.

Aggregation function (Equation 3):

$$\mathcal{F}_{\text{agg}}(\mathbf{c}) = \left((m-1) - \sum_{i=1}^{m-1} c_i,\ c_m \right) \tag{Eq. 3}$$

What it computes:

- $(m-1) - \sum_{i=1}^{m-1} c_i$: the number of POIs *not yet seen*. Since we want to *minimize* this (see as many as possible), subtracting the count of seen POIs from the total gives us a quantity to minimize.
- c_m: path length, unchanged.

Numerical Example

With $q = 4$ POIs, suppose a path has seen POIs 1, 3, and 4 (but not 2): $c_1 = 1, c_2 = 0, c_3 = 1, c_4 = 1$, and path length $c_5 = 20$.
Aggregated cost: $(4 - (1+0+1+1),\ 20) = (1, 20)$.
One POI remains unseen, and the path is 20 units long.

3.4.3 Example 3: Route Planning with Road Types (Equations 4–5)

Setting: Plan a route on a road network with paved and unpaved roads. Minimize overall path length while avoiding long consecutive stretches of unpaved road.

Hidden objectives: $m = 3$: accumulated road length (g), current consecutive unpaved length (g_{con}), and maximum consecutive unpaved length ($g_{\max}$).

Edge cost encoding: Each edge e provides (ℓ, Type), where ℓ is the road length and $\text{Type} \in \{0, 1\}$ indicates unpaved (0) or paved (1).

Extension function (Equation 4):

$$\mathcal{F}_{\text{ext}}\big((g, g_{\text{con}}, g_{\max}),\ (\ell, \text{Type})\big) = \begin{cases} \big(g+\ell,\ g_{\text{con}}+\ell,\ \max(g_{\text{con}}+\ell, g_{\max})\big) & \text{if Type} = 0, \\ \big(g+\ell,\ 0,\ g_{\max}\big) & \text{if Type} = 1. \end{cases} \tag{Eq. 4}$$

What it computes (case by case):

- **Unpaved edge** (Type $= 0$):
  - $g + \ell$: total path length increases.
  - $g_{\text{con}} + \ell$: consecutive unpaved stretch grows.
  - $\max(g_{\text{con}} + \ell, g_{\max})$: update the worst (longest) consecutive unpaved stretch.
- **Paved edge** (Type $= 1$):
  - $g + \ell$: total path length increases.
  - 0: consecutive unpaved counter *resets to zero* (the streak is broken).
  - $g_{\max}$: worst unpaved stretch stays as recorded.

Aggregation function (Equation 5):

$$\mathcal{F}_{\text{agg}}(g, g_{\text{con}}, g_{\max}) = (g,\ g_{\max}). \tag{Eq. 5}$$

What it computes: The solution objectives are simply the total path length and the maximum consecutive unpaved road length. The "current consecutive unpaved" counter g_{con} is a hidden objective needed *during* the search but not part of the final solution cost.

Non-Monotone Hidden Objectives

Notice that g_{con} is *not* monotonically non-decreasing: a paved edge resets it to 0. This means $\mathcal{F}_{\text{ext}}$ is only monotone with respect to g and $g_{\max}$, not g_{con}. The paper discusses that monotonicity need only hold for the *solution objectives* (after aggregation), and since g_{con} is dropped by $\mathcal{F}_{\text{agg}}$, the approach still works. This is an important subtlety.

Numerical Example

Consider a path through edges with these properties:

Edge	Length ℓ	Type	Description
e_1	5	0 (unpaved)	Start on dirt road
e_2	3	0 (unpaved)	Still on dirt
e_3	10	1 (paved)	Hit the highway
e_4	4	0 (unpaved)	Back to dirt

Step by step:

$$\begin{aligned}
\text{After } e_1: &\quad (5,\ 5,\ \max(5,0)) = (5,\ 5,\ 5)\\
\text{After } e_2: &\quad (8,\ 8,\ \max(8,5)) = (8,\ 8,\ 8)\\
\text{After } e_3: &\quad (18,\ 0,\ 8) \quad \text{(paved: reset } g_{\text{con}})\\
\text{After } e_4: &\quad (22,\ 4,\ \max(4,8)) = (22,\ 4,\ 8)
\end{aligned}$$

Aggregation: $\mathcal{F}_{\text{agg}}(22, 4, 8) = (22, 8)$.
Solution: total length 22, worst consecutive unpaved stretch was 8 (from e_1 and e_2).

4 Algorithms

This section explains the multi-objective A* algorithm and the minimal changes needed to support objective aggregation.

4.1 Algorithm 1: MOS-A* (The Base Algorithm)

The base algorithm is a multi-objective version of A*. We explain it line by line.

Key Difference from Single-Objective A*

In standard A*, each node has a single best g-value. In MOS-A*, each node can have **multiple non-dominated** g-values (one for each Pareto-optimal path to that node). The algorithm maintains a "label" at each node: a set of non-dominated cost vectors representing the best paths found so far.

Data structures:

- $\mathbf{g}(n) \in \mathbb{R}^m_{\geq 0}$: the cost vector of the path from the start to node n (in the hidden objective space).
- $\mathbf{f}(n) = \mathbf{g}(n) + \mathbf{h}(v(n))$: the estimated total cost (where $\mathbf{h}$ is the heuristic at the node's vertex $v(n)$).
- OPEN: priority queue of unexpanded nodes, ordered lexicographically by $\mathbf{f}$.
- SOLS: set of goal nodes reached (solutions found so far).
- CLOSED: set of already-expanded nodes.

Line-by-line explanation of Algorithm 1:

Line 1: Create the root node n_{root} at the start vertex v_s, with $\mathbf{g} = \mathbf{0}$ and $\mathbf{f} = \mathbf{h}(v_s)$.

Line 2: Insert the root into OPEN. Initialize SOLS and CLOSED as empty sets.

Line 3: Main loop — keep searching while OPEN is non-empty.

Line 4: Extract the node n with the **lexicographically smallest f-value** from OPEN (function `get_best_node`).

Lines 5–6: Dominance pruning. If n's cost is dominated by an existing solution (`is_dominated_sol`) or by another path to the same vertex already in CLOSED (`is_dominated_path`), skip n and go back to Line 3.

Lines 7–9: If n is at the goal vertex, add it to SOLS and skip expansion (solutions are not expanded further).

Lines 10–14: Expansion. For each neighbor v' of n's vertex:

- Create a new child node n' at v' (Line 11).
- Compute $\mathbf{g}(n') = \texttt{compute\_cost}(\mathbf{g}(n),\ \mathbf{c}(v(n), v'))$ (Line 12). In the standard algorithm, this is just $\mathbf{g}(n) + \mathbf{c}(e)$.
- Compute $\mathbf{f}(n') = \mathbf{g}(n') + \mathbf{h}(v')$ (Line 13).
- Insert n' into OPEN (Line 14).

Line 15: Move n to CLOSED.

Line 16: When OPEN is exhausted, return SOLS as the POF.

Simple MOS-A* trace

Consider a graph with 3 vertices $\{S, A, G\}$ (start S, goal G), $d = 2$ objectives, and edges:

Edge	Cost (c_1, c_2)
(S, A)	$(1, 5)$
(S, G)	$(4, 2)$
(A, G)	$(2, 1)$

Using $\mathbf{h} = \mathbf{0}$ (no heuristic) for simplicity:

1. Initialize: n_S with $\mathbf{g} = (0, 0)$, $\mathbf{f} = (0, 0)$. OPEN $= \{n_S\}$.
2. Extract n_S. Expand:
   - n_A: $\mathbf{g} = (1, 5)$, $\mathbf{f} = (1, 5)$.
   - n_{G1}: $\mathbf{g} = (4, 2)$, $\mathbf{f} = (4, 2)$.

   OPEN $= \{n_A, n_{G1}\}$.
3. Extract n_A ($\mathbf{f} = (1, 5)$ is lex-smaller than $(4, 2)$ since $1 < 4$). Expand:
   - n_{G2}: $\mathbf{g} = (1 + 2, 5 + 1) = (3, 6)$, $\mathbf{f} = (3, 6)$.

   OPEN $= \{n_{G2}, n_{G1}\}$.
4. Extract n_{G2} ($\mathbf{f} = (3, 6)$ vs. $(4, 2)$: $3 < 4$). Node n_{G2} is a solution. Add to SOLS. Is $(3, 6)$ dominated by $(4, 2)$? No ($3 < 4$ but $6 > 2$).
5. Extract n_{G1} ($\mathbf{f} = (4, 2)$). Is $(4, 2)$ dominated by $(3, 6)$? No ($4 > 3$ but $2 < 6$). Add to SOLS.
6. OPEN empty. Return SOLS $= \{(3, 6), (4, 2)\}$.

The POF contains two solutions: the short-but-costly path $S \to A \to G$ with cost $(3, 6)$, and the longer-but-cheaper path $S \to G$ with cost $(4, 2)$.

4.2 Algorithm 2: Core Subroutines

Algorithm 2 in the paper defines four helper functions. We explain each one.

4.2.1 `get_best_node`(OPEN)

What it does: Returns the node in OPEN with the lexicographically smallest $\mathbf{f}$-value.

In the standard algorithm, the ordering is by $\mathbf{f}(n)$ (the hidden objective cost estimate). In the **aggregation-extended** version, the ordering is by $\mathcal{F}_{\text{agg}}(\mathbf{f}(n))$ — the solution-level cost estimate.

Why this change matters: Ordering by the aggregated cost in $\mathbb{R}^k$ (with $k < m$) leads to better dimensionality reduction in the priority queue. Standard MOS tools exploit the smaller dimensionality to prune more effectively.

4.2.2 `is_dominated_sol`(n', SOLS)

What it does: Checks if the new node n' is dominated by any existing solution in SOLS. If so, n' can be safely discarded — any path through n' cannot lead to a non-dominated solution.

In the standard algorithm, dominance is checked on $\mathbf{f}(n)$. In the aggregation-extended version, dominance is checked on $\mathcal{F}_{\text{agg}}(\mathbf{f}(n))$ — comparing solution-level costs.

4.2.3 `is_dominated_path`(n', CLOSED)

What it does: Checks if there is already an expanded node in CLOSED at the same vertex as n', with a dominating $\mathbf{g}$-value. If so, n' represents a path to the same vertex that is no better than one already explored.

Important: This function always compares the *hidden* costs $\mathbf{g}$, not the aggregated costs. This is because two paths might have different hidden costs but the same aggregated cost, and we need to keep both to ensure correctness when extending to different successors.

4.2.4 `compute_cost`($\mathbf{g}_{\text{parent}}, \mathbf{c}$)

What it does: Computes the $\mathbf{g}$-value of a child node from its parent's $\mathbf{g}$-value and the edge cost.

In the standard algorithm: $\mathbf{g}_{\text{parent}} + \mathbf{c}$ (addition).

In the aggregation-extended version: $\mathcal{F}_{\text{ext}}(\mathbf{g}_{\text{parent}}, \mathbf{c})$ (the path-cost extension function).

4.3 From Exact to Approximate MOS

As discussed in Section 1.4, computing the exact POF can be extremely expensive. The paper describes a simple and general recipe for turning any exact MOS algorithm into an approximate one.

The recipe (Section III-C-2 of the paper): Given an approximation factor ε, an **ε-approximate POF** Π^*_ε is a set of solutions such that every path in the true POF Π^* is ε-dominated by a path in Π^*_ε.

To compute Π^*_ε, one simply replaces the domination test $\preceq$ used in solution domination (i.e., Line 1 of function `is_dominated_sol` in Algorithm 2) with the approximate domination test $\preceq_\varepsilon$.

Approximate MOS Is a One-Line Change

Any exact MOS algorithm can be converted to an approximate one by replacing the dominance check in `is_dominated_sol` with ε-dominance. No other changes are needed. The result is a smaller set of solutions that still "covers" the entire true POF within the approximation tolerance.

Intuition: With exact dominance, a new solution is only pruned if an existing solution is at least as good on *every* objective. With ε-dominance, a new solution is pruned if an existing solution is within a factor of $(1 + \varepsilon_i)$ on each objective. This allows more aggressive pruning, yielding a smaller (but still representative) approximate POF.

4.4 The Straw Man Approach vs. Objective Aggregation

The paper contrasts two approaches to computing the POF w.r.t. $\mathcal{F}_{\text{agg}}$:

The straw man (m, m)-approach:

1. Run MOS with all m hidden objectives to compute the full POF Π^* in $\mathbb{R}^m$.
2. For each solution $\pi \in \Pi^*$, compute $\mathbf{c}_{\mathcal{F}_{\text{agg}}}(\pi)$.
3. Return the non-dominated solutions among the aggregated costs.

The problem: The full POF Π^* can be *enormous* because it lives in m-dimensional space. Computing it is the bottleneck.

The paper's (k, m)-approach: Modify the MOS algorithm to:

1. Use $\mathcal{F}_{\text{ext}}$ in `compute_cost` (track m hidden objectives).
2. Order OPEN and check solution-dominance using $\mathcal{F}_{\text{agg}}$ (operate in k-dimensional solution space).
3. Check path-dominance using the hidden $\mathbf{g}$ values (preserve correctness).

Why the (k, m)-Approach Is Faster

The key insight is that even though the algorithm *tracks* m hidden objectives during search, it *prunes* based on k solution objectives. Since $k < m$, more solutions appear "equivalent" and can be pruned. The resulting POF has far fewer elements, and the search explores far fewer nodes.
Think of it this way: in the obstacle-uncertainty problem with $q = 12$ obstacles, the straw

man works in $m = 13$ dimensions. The paper's approach works in $k = 2$ dimensions (risk + length). Two paths with different per-obstacle risk profiles but the same total risk and length are equivalent in the solution space — the straw man would keep both, but the paper's approach prunes one.

4.5 Summary of Algorithmic Changes

The changes from the standard MOS-A* to the aggregation-extended version are surprisingly small. Only **three functions** need modification:

Function	Standard MOS	With Aggregation
`get_best_node`	Order by $\mathbf{f}(n)$	Order by $\mathcal{F}_{\text{agg}}(\mathbf{f}(n))$
`is_dominated_sol`	Compare $\mathbf{f}(n)$	Compare $\mathcal{F}_{\text{agg}}(\mathbf{f}(n))$
`compute_cost`	$\mathbf{g} + \mathbf{c}$	$\mathcal{F}_{\text{ext}}(\mathbf{g}, \mathbf{c})$
`is_dominated_path`	Unchanged (always uses hidden $\mathbf{g}$)	

The Heuristic Must Match the Aggregated Cost

When ordering by $\mathcal{F}_{\text{agg}}(\mathbf{f}(n))$, the heuristic $\mathbf{h}$ should be computed with respect to the *aggregated* cost, not the hidden cost. The paper uses graph-distance heuristics (shortest-path distance) for the path-length objective, which is straightforward.

5 Theoretical Analysis: Theorem 1

The paper's main theoretical result establishes that the (k, m)-approach is *correct* — it produces the same POF as the straw man approach.

5.1 Statement of Theorem 1

Theorem 5.1 (Correctness of objective aggregation)**.** *Let **ALG** be some MOS algorithm and consider a MOS problem with k solution and m hidden objectives for some $k < m$, respectively, and with monotonically non-decreasing path extension and objective-aggregation functions $\mathcal{F}_{\text{ext}}$ and $\mathcal{F}_{\text{agg}}$. If (m, m)-**ALG** returns a POF to the MOS problem, then (k, m)-**ALG** (using the aggregation-extended subroutines) also returns a POF to the MOS problem.*

What it says in plain English: If the base MOS algorithm is correct (i.e., it computes the full POF when given all objectives), then the modified version with aggregation is *also* correct: it computes the POF with respect to the solution objectives.

In other words, the three small algorithmic changes described above are *sufficient* to correctly handle aggregation — no fundamental redesign of the algorithm is needed.

Symbol breakdown:

- (m, m)-**ALG**: the straw man approach — run the algorithm with m hidden objectives as the solution objectives (no aggregation).
- (k, m)-**ALG**: the paper's approach — track m hidden objectives but optimize k solution objectives (with aggregation).

- $\mathcal{F}_{\text{ext}}$: the path-cost extension function (how to accumulate costs).
- $\mathcal{F}_{\text{agg}}$: the objective-aggregation function (how to compute solution costs from hidden costs).
- "Monotonically non-decreasing": the requirement that both functions preserve the dominance ordering.

5.2 Proof Sketch

The proof proceeds in four main steps. We explain each intuitively.

5.2.1 Step 1: Completeness (no paths are lost)

The lexicographic key used to sort OPEN does not affect which paths are explored — it only affects the *order* of exploration. The search terminates only when OPEN is empty, so all reachable paths will eventually be generated. The function `is_dominated_path` is identical in both versions (it uses the hidden $\mathbf{g}$ values), so the same set of non-dominated paths survives pruning.

Consequence: (k, m)-**ALG** generates at least as many un-dominated paths as (m, m)-**ALG**.

5.2.2 Step 2: Path dominance is unchanged

The `is_dominated_path` function compares hidden $\mathbf{g}$-values. Since this function is identical in both versions, a path that survives in (m, m)-**ALG** also survives in (k, m)-**ALG**.

5.2.3 Step 3: Monotonicity preserves encounter order

Since both $\mathcal{F}_{\text{ext}}$ and $\mathcal{F}_{\text{agg}}$ are monotonically non-decreasing, their composition $\mathbf{c}_{\mathcal{F}_{\text{agg}}}(\cdot) = \mathcal{F}_{\text{agg}}(\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\cdot))$ is also monotonically non-decreasing. This means the aggregated $\mathbf{g}$-values are non-decreasing, so solutions in the POF are encountered before non-optimal solutions.

Intuition: Monotonicity ensures that a "good" partial path remains "good" after extension. If a partial path has lower aggregated cost than another, it will continue to have lower aggregated cost after extending both by the same edge. This means the best solutions are found first (or at least, no worse solution is found first), which is crucial for correct pruning.

5.2.4 Step 4: Solution dominance filters correctly

The function `is_dominated_sol` uses $\mathcal{F}_{\text{agg}}$ in the (k, m) version. Since optimal solutions are found before non-optimal ones (Step 3), every non-optimal solution will be correctly identified as dominated and discarded.

The Proof's Key Insight

The proof shows that the only change that "matters" is how dominance is measured at the solution level. By using $\mathcal{F}_{\text{agg}}$ for solution comparisons and keeping the hidden-level path comparisons unchanged, the algorithm correctly computes the POF in the k-dimensional solution space while tracking the necessary detail in the m-dimensional hidden space.

5.3 Relaxing Monotonicity of $\mathcal{F}_{\text{ext}}$

The paper notes an interesting subtlety: the monotonicity requirement on $\mathcal{F}_{\text{ext}}$ can be relaxed. What matters is that monotonicity holds for the *solution objectives*, i.e., the composition $\mathbf{c}_{\mathcal{F}_{\text{agg}}}(\pi) := \mathcal{F}_{\text{agg}}(\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi))$ must be monotonically non-decreasing. The extension function itself need not be monotone in all components.

This is exactly the situation in the route-planning example (Section 3.4): g_{con} can decrease (reset to 0) when hitting a paved edge, so $\mathcal{F}_{\text{ext}}$ is not monotone in all components. But after aggregation by $\mathcal{F}_{\text{agg}}$ (which drops g_{con}), the solution objectives $(g, g_{\max})$ are monotonically non-decreasing.

6 Representative Applications in Detail

The paper demonstrates the framework on four robotics scenarios. We explain the key aspects of each.

6.1 Planning Under Obstacle Uncertainty (OU)

6.1.1 The Shadow Model

The paper formalizes obstacle uncertainty through the notion of *shadows*, introduced by Axelrod et al. [2]. We present the full chain of definitions from the paper.

Definition 6.1 (Shadows — Definition 1 in the paper)**.** For each obstacle $O_i \in O$, a sequence of volumes called **shadows** $S(O_i) = \{S_0^i, S_1^i, \ldots\}$ is defined, where $\forall\, i, j : S_j^i \subseteq \mathcal{W}$ and $S_j^i \subseteq S_{j+1}^i$ (each shadow is contained in the workspace, and shadows are nested).

Definition 6.2 (Shadow risk bound $r_{\max}$)**.** A function $r_{\max} : \mathcal{S} \to [0, 1]$ associates each shadow with an upper bound on the collision probability with its underlying obstacle. For every configuration x:

$$r_{\max}(S_j^i) \;\geq\; \text{Prob}\big[\,\text{Shape}(x) \cap O_i \neq \emptyset \;\big|\; \text{Shape}(x) \cap S_j^i \neq \emptyset\,\big].$$

Risk increases monotonically for smaller (inner) shadows: $\forall\, i, j, j'$ s.t. $j < j'$ it holds that $r_{\max}(S_j^i) > r_{\max}(S_{j'}^i)$.

Intuition: Think of shadows like contour lines around an obstacle. The innermost contour (closest to the true obstacle position) has the highest collision probability. As you move outward, the probability decreases. A robot passing through the outermost shadow has a small chance of collision; passing through the innermost shadow has a high chance.

Definition 6.3 (Shadow risk at a configuration)**.** The risk of the robot at configuration x intersecting shadow S_j^i of obstacle O_i is:

$$\text{risk}(x, S_j^i) := r_{\max}(S_j^i).$$

Definition 6.4 (Effective shadow)**.** When the robot is at configuration x, it may intersect multiple shadows of the same obstacle simultaneously. The **effective shadow** of obstacle O_i at configuration x is the smallest (innermost) shadow intersected:

$$S_{\text{eff}}^i(x) := S^i_{\arg\min_j \{j \,|\, \text{Shape}(x) \cap S_j^i \neq \emptyset\}}.$$

The effective shadow gives the tightest (highest) risk bound.

Definition 6.5 (Obstacle risk at a configuration)**.** The risk of the robot at configuration x with respect to obstacle O_i is the risk of its effective shadow:

$$\text{risk}(x, O_i) := \text{risk}(x, S_{\text{eff}}^i(x)).$$

Definition 6.6 (Path obstacle risk)**.** A robot's path $\pi : [0, 1] \to \mathcal{X}$ is a continuous mapping from the unit interval into configuration space. The **path obstacle risk** is the maximal obstacle risk along the path:

$$\text{risk}(\pi, O_i) := \max_{\alpha \in [0,1]} \{\text{risk}(\pi(\alpha), O_i)\}.$$

This is why the extension function uses max for per-obstacle risks: the path's risk for each obstacle is determined by the worst-case point along the path.

Definition 6.7 (Total path risk)**.** Assuming that the risk of intersecting different obstacles is *mutually independent*, the total risk of a path π is:

$$\text{risk}(\pi) := 1 - \prod_{O_i \in O} (1 - \text{risk}(\pi, O_i)).$$

This is exactly what $\mathcal{F}_{\text{agg}}$ computes (Equation 2).

Numerical Example

A robot's path passes through the following effective shadows:

Obstacle	Per-obstacle risk c_i
O_1	0.3
O_2	0.0 (path avoids O_2's shadows entirely)
O_3	0.15

Total risk: $1 - (1 - 0.3)(1 - 0.0)(1 - 0.15) = 1 - 0.7 \times 1.0 \times 0.85 = 1 - 0.595 = 0.405$.
The path has a 40.5% chance of colliding with at least one obstacle.

6.1.2 Why the Straw Man Is Slow

In the obstacle-uncertainty scenario:

- $m = q + 1$ (one risk per obstacle + path length).
- $k = 2$ (total risk + path length).

The straw man (m, m)-approach computes the POF in $(q + 1)$-dimensional space. With $q = 12$ obstacles, this means tracking all 2^{12}-like combinations of per-obstacle risk levels — the POF can have thousands or millions of elements.

The paper's (k, m)-approach computes the POF in 2D (risk vs. length). Two paths with different per-obstacle risk profiles but the same total risk and length are considered equivalent — the smaller POF leads to dramatically faster search (up to 5000× speedup in experiments).

6.2 Inspection Planning

A robot explores an environment with q points of interest (POIs). At each step, it may "see" (cover) one or more POIs. The goals are to maximize coverage (see as many POIs as possible) and minimize path length.

In the objective-aggregation framework:

- Hidden objectives: $m = q + 1$ (one binary per POI + path length).
- Solution objectives: $k = 2$ (number unseen + path length).
- $\mathcal{F}_{\text{ext}}$: uses max for POI indicators (once seen = always seen).
- $\mathcal{F}_{\text{agg}}$: counts unseen POIs + passes through path length (Equation 3).

6.3 Route Planning with Road Types

A vehicle plans a route on a road network with paved and unpaved roads. Goals: minimize total distance and minimize the worst consecutive unpaved stretch.

In the objective-aggregation framework:

- Hidden objectives: $m = 3$ (g, g_{con}, $g_{\max}$).
- Solution objectives: $k = 2$ (g, $g_{\max}$).
- $\mathcal{F}_{\text{ext}}$: Equation (4) (tracks current and maximum consecutive unpaved).
- $\mathcal{F}_{\text{agg}}$: Equation (5) (drops the current consecutive counter).

The subtlety here is that g_{con} is non-monotone, but since it is dropped by $\mathcal{F}_{\text{agg}}$, the framework still applies.

7 Experimental Results

The paper evaluates the approach on four scenarios using NAMOA-dr as the base MOS algorithm, comparing:

- **Baseline**: the straw man (m, m)-approach.
- **ObjAgg**: the proposed (k, m)-approach.

7.1 Key Experimental Findings

7.1.1 Point Robot Navigation (OU)

- Environment: 2D plane with rectangular obstacles, $q = 8$ or $q = 12$ obstacles, 30 shadows per obstacle.
- Hidden objectives: $m = 9$ or $m = 13$ (per-obstacle risks + length).
- Solution objectives: $k = 2$ (total risk + length).
- **Result:** ObjAgg achieves speedups of 4× to 265× over Baseline, with larger speedups for larger ε and more obstacles.
- With 12 obstacles, speedups reach up to 5000×.

Why More Obstacles = More Speedup

With more obstacles, m grows while k stays fixed at 2. The gap between m-dimensional and k-dimensional POFs widens, so the aggregation-based pruning becomes increasingly effective. The Baseline must track an exponentially growing number of Pareto-optimal per-obstacle risk combinations, while ObjAgg collapses them all to a single total risk value.

7.1.2 Planar Manipulator (OU)

- A 5-link planar arm reaching into a cluttered refrigerator shelf.
- $q = 12$ obstacles, $m = 13$ hidden objectives.
- **Result:** ObjAgg takes 152 seconds ($\varepsilon = 0$) vs. Baseline hitting the 1500-second timeout for all ε values. Speedups of 10×–19×.

7.1.3 CRISP Surgical Robot (OU)

- A continuum robot navigating inside a lung cavity (from real patient CT scan).
- $m = 4$ hidden objectives (3 obstacle regions + length).
- **Result:** Speedups of roughly 5×. The smaller speedup (compared to point robot) is because $m = 4$ is not much larger than $k = 2$.

7.1.4 Route Planning with Road Types

- OpenStreetMap road network of Boulder Foothills, Colorado.
- $m = 3$ hidden objectives, $k = 2$ solution objectives.
- **Key insight:** g_{con} is non-monotone, so the lexicographic order used in Baseline's OPEN is sensitive to the ordering of hidden objectives. Different orderings (CLM, MLC, MCL) produce runtimes varying by up to 10×.
- **Result:** ObjAgg achieves 2×–10× speedups and, critically, is *insensitive* to the ordering of hidden objectives.

Robustness to Objective Ordering

Baseline is highly sensitive to the lexicographic order used when hidden objectives are non-monotone. Some orderings cause many non-optimal solutions to enter SOLS, wasting computation. ObjAgg avoids this problem because it orders by the *aggregated* solution cost, which does not depend on the ordering of hidden objectives.

8 Putting It All Together: The Big Picture

Let us step back and summarize the paper's contribution in the context of the broader multi-objective search landscape.

8.1 The Problem

Many real-world robotic planning problems involve objectives that *interact* in complex ways:

- Collision risks from different obstacles combine multiplicatively (not additively).
- Some objectives (like "consecutive unpaved road length") are non-monotone and only meaningful at the end of the path.
- The number of "raw" objectives (m) can be large (one per obstacle or POI), making standard MOS intractable.

Standard MOS algorithms cannot handle these cases because they assume:

1. Additive path costs.
2. All objectives are directly optimized (no aggregation).

8.2 The Solution

The paper introduces two functions — $\mathcal{F}_{\text{ext}}$ and $\mathcal{F}_{\text{agg}}$ — that generalize the standard MOS formulation:

- $\mathcal{F}_{\text{ext}}$ generalizes path-cost computation to handle non-additive accumulation.
- $\mathcal{F}_{\text{agg}}$ reduces the dimensionality of the optimization problem by mapping m hidden objectives to k solution objectives.

The beauty of this approach is its **minimality**: only three subroutines of the existing MOS algorithm need modification. Any existing MOS algorithm (NAMOA-dr, A*pex, etc.) can be upgraded with these changes.

8.3 The Guarantee

Theorem 1 proves that the modified algorithm is *correct*: it computes the exact same POF (w.r.t. solution objectives) as the straw man approach, but typically much faster.

The correctness relies on the **monotonicity** of $\mathcal{F}_{\text{ext}}$ and $\mathcal{F}_{\text{agg}}$, which ensures that the dominance-based pruning remains valid.

8.4 The Practical Impact

Empirical results show speedups of up to 5000× over the baseline approach, with larger gains when:

- The gap between m (hidden) and k (solution) objectives is large.
- The approximation factor ε is larger (more aggressive pruning of near-equivalent solutions).
- The number of obstacles/POIs is large.

9 Summary of Notation

For quick reference, here is a table of all major symbols used in the paper:

Symbol	Type / Dimensions	Meaning
	Graph and Search	
$G = (V, E, \mathbf{c})$	MOS Graph	Graph with vertices, edges, and cost function
V	Finite set	Set of vertices (locations/states)
$E \subseteq V \times V$	Finite set	Set of edges (transitions)
$\mathbf{c} : E \to \mathbb{R}^d_{\geq 0}$	Function	Edge cost function (d-dimensional)
d	Scalar (integer)	Dimension of edge cost vectors
v_s, v_g	Vertices	Start and goal vertices
π	Sequence of vertices	A path from v_s to v_g
$\mathbf{c}(\pi)$	Vector $\in \mathbb{R}^d_{\geq 0}$	Cost of path π (standard: sum of edge costs)
	Multi-Objective Concepts	
$\mathbf{p} \preceq \mathbf{q}$	Relation	$\mathbf{p}$ dominates $\mathbf{q}$ ($p_i \leq q_i$ for all i)
$\mathbf{p} \prec_{\text{lex}} \mathbf{q}$	Relation	$\mathbf{p}$ is lexicographically smaller than $\mathbf{q}$
$\mathbf{p} \preceq_\varepsilon \mathbf{q}$	Relation	$\mathbf{p}$ ε-dominates $\mathbf{q}$
ε	Vector $\in \mathbb{R}^d_{\geq 0}$	Approximation factor
Π^*	Set of paths	Pareto-optimal frontier
Π^*_ε	Set of paths	ε-approximate Pareto-optimal frontier
	Objective Aggregation Framework	
m	Scalar (integer)	Hidden objective dimensionality
k	Scalar (integer)	Solution objective dimensionality ($k \leq m$)
$\mathcal{F}_{\text{ext}}$	$\mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^m$	Path-cost extension function
$\mathcal{F}_{\text{agg}}$	$\mathbb{R}^m \to \mathbb{R}^k$	Objective-aggregation function
$\mathbf{c}_{\mathcal{F}_{\text{ext}}}(\pi)$	Vector $\in \mathbb{R}^m$	Extended (hidden) cost of path π
$\mathbf{c}_{\mathcal{F}_{\text{agg}}}(\pi)$	Vector $\in \mathbb{R}^k$	Solution cost of path π
$\Pi^*_{\mathcal{F}_{\text{agg}}}$	Set of paths	POF w.r.t. aggregation function $\mathcal{F}_{\text{agg}}$
	Algorithm Data Structures	
$\mathbf{g}(n)$	Vector $\in \mathbb{R}^m_{\geq 0}$	Hidden cost from start to node n
$\mathbf{h}(v)$	Vector $\in \mathbb{R}^m_{\geq 0}$	Heuristic estimate from vertex v to goal
$\mathbf{f}(n)$	Vector $\in \mathbb{R}^m_{\geq 0}$	Estimated total cost: $\mathbf{g}(n) + \mathbf{h}(v(n))$
OPEN	Priority queue	Nodes to be expanded
CLOSED	Set of nodes	Already-expanded nodes
SOLS	Set of nodes	Solutions found so far
	Application-Specific Notation	
q	Scalar (integer)	Number of obstacles or POIs
O_i	Obstacle	The i-th obstacle
$S(O_i)$	Set of volumes	Shadows of obstacle O_i
S^i_j	Volume	The j-th shadow of obstacle O_i
$r_{\max}(S^i_j)$	Scalar $\in [0, 1]$	Max collision probability of shadow S^i_j
$\text{risk}(\pi, O_i)$	Scalar $\in [0, 1]$	Risk of path π w.r.t. obstacle O_i
$\text{risk}(\pi)$	Scalar $\in [0, 1]$	Total collision risk of path π
g_{con}	Scalar ≥ 0	Current consecutive unpaved road length
$g_{\max}$	Scalar ≥ 0	Maximum consecutive unpaved road length
(k, m)-**ALG**$_\varepsilon$	Algorithm label	MOS algorithm with k solution, m hidden objectives, approximation ε