Technical Companion to

Tightest Admissible Shortest Path

A Self-Contained Guide for CS Undergraduates

Companion to the paper by Eyal Weiss, Ariel Felner, and Gal A. Kaminka
(ICAPS 2024)

Abstract

This document explains every equation, definition, theorem, and algorithm in the paper "Tightest Admissible Shortest Path" in detail, assuming only undergraduate-level knowledge of graph search algorithms (Dijkstra's algorithm, A*, admissible heuristics) and basic graph theory. No prior knowledge of estimated weighted digraphs, cost estimator functions, or the specific contributions of this paper 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: Shortest Paths, Heuristics, and Admissibility

1.1 What Is the Shortest Path Problem?

The **shortest path problem** is one of the most fundamental problems in computer science and artificial intelligence.

Key Idea

Given a graph where edges have costs (weights), find the path from a *source* node to a *goal* node whose total edge cost is minimized.

Formally, a **weighted directed graph (digraph)** is a tuple (V, E, c) where:

- V is a finite set of **nodes** (also called vertices).
- $E \subseteq V \times V$ is a set of **directed edges**. An edge $e = (v_i, v_j) \in E$ means there is a directed connection from v_i to v_j. Unlike undirected graphs, $(v_i, v_j) \in E$ does *not* imply $(v_j, v_i) \in E$.
- $c : E \to \mathbb{R}^+$ is a **cost function** that assigns a non-negative real number to every edge. The cost $c(e)$ represents the expense of traversing edge e (e.g., travel time, fuel consumption, monetary cost).

A **path** $\pi = \langle e_1, e_2, \ldots, e_n \rangle$ from v_i to v_j is a sequence of edges where each edge starts where the previous one ended. Formally, $e_k = (v_{q_k}, v_{q_{k+1}})$ for $k \in [1, n]$, with $v_i = v_{q_1}$ and $v_j = v_{q_{n+1}}$.

The **cost of a path** is the sum of the costs of its edges:

$$c(\pi) := \sum_{k=1}^{n} c(e_k).$$

The **Goal-Directed Single-Source Shortest Path** (GDS^3P) problem asks: given a source node v_s and a set of goal nodes $V_g \subset V$, find a path π from v_s to some $v \in V_g$ that minimizes $c(\pi)$. The cost of an optimal solution is denoted C^*.

Numerical Example

Consider a graph with nodes $\{A, B, C, D\}$ and edges:

Edge	Cost $c(e)$
(A, B)	3
(A, C)	5
(B, D)	4
(C, D)	1

With source $v_s = A$ and goal $V_g = \{D\}$:

- Path $\pi_1 = \langle (A, B), (B, D) \rangle$: cost $= 3 + 4 = 7$.
- Path $\pi_2 = \langle (A, C), (C, D) \rangle$: cost $= 5 + 1 = 6$.

The shortest path is π_2 with $C^* = 6$.

1.2 Algorithms: Dijkstra's and Uniform Cost Search

Dijkstra's algorithm (and its close relative **Uniform Cost Search**, or UCS) solve the shortest path problem by maintaining a priority queue (called OPEN) of nodes to explore, ordered by the cost of the best known path to each node.

At each step, the algorithm:

1. Removes the node n with the smallest known path cost $g(n)$ from OPEN.
2. If n is a goal node, returns the path.
3. Otherwise, expands n: for each successor s of n, if the path through n to s is cheaper than the best previously known path to s, updates s's cost and adds/updates it in OPEN.

A* extends UCS by using a *heuristic function* $h(n)$ that estimates the remaining cost from n to the goal, ordering OPEN by $f(n) = g(n) + h(n)$ instead of $g(n)$ alone.

1.3 Admissibility and Suboptimality Bounds

Definition 1.1 (B-admissible shortest path)**.** A solution π for a GDS^3P problem is said to be a **B-admissible shortest path** if its cost is bounded by a suboptimality factor B:

$$c(\pi) \leq C^* \times B. \tag{1}$$

What it says in plain English: The solution π may not be optimal, but its cost is guaranteed to be at most B times the cost of the true shortest path.

Symbol-by-symbol breakdown:

- $c(\pi)$: the total cost of the solution path π.
- C^*: the cost of the true optimal (shortest) path.
- $B \geq 1$: the **suboptimality factor** (also called the "admissibility factor"). If $B = 1$, the solution is optimal. If $B = 1.5$, the solution is at most 50% more expensive than optimal.

Key Idea

B-admissibility is a quality guarantee: it tells you the *worst case* ratio between the cost of your solution and the cost of the true shortest path. Smaller B means a tighter (better) guarantee.

Numerical Example

If $C^* = 10$ (the shortest path costs 10):

- A path with cost 10 is 1-admissible (optimal).
- A path with cost 12 is $12/10 = 1.2$-admissible (at most 20% over optimal).
- A path with cost 15 is $15/10 = 1.5$-admissible (at most 50% over optimal).

Admissibility Requires Knowing C^*

In the standard setting, determining whether a path is B-admissible requires knowing the true optimal cost C^* and the true cost $c(\pi)$—both of which depend on exact edge weights. The key challenge of this paper arises when edge weights are *not* exactly known, but are instead bounded by estimates. In that case, we cannot compute $c(\pi)$ or C^* exactly, and the notion of B-admissibility must be generalized.

2 The Problem Setting: Estimated Weighted Digraphs

2.1 Why Edge Weights Are Not Always Known

In many real-world applications, the exact cost of an edge is not available for free. It may require expensive computation:

- **Drone flight planning**: The energy consumption of a flight segment depends on wind, weather, and vehicle characteristics. A rough estimate (from a database of distances and speed limits) is cheap; a more precise estimate (using wind forecasts and aerodynamic models) is expensive.
- **Robot motion planning**: Checking whether a robot configuration is collision-free and computing the exact traversal cost requires expensive geometric and kinematic computations.
- **Navigation**: Quick driving-time estimates use simple distance/speed-limit lookups, while more accurate estimates query traffic data, road conditions, and vehicle-specific parameters.

The key insight is that for each edge, we can obtain *multiple estimates* of its true weight, at increasing levels of accuracy and computational cost. Cheaper estimates give looser bounds; more expensive estimates give tighter bounds.

2.2 Cost Estimators Function Θ

Definition 2.1 (Cost estimators function)**.** For a set of edges E and a cost function $c : E \to \mathbb{R}^+$, a **cost estimators function** Θ maps every edge $e \in E$ to a finite and non-empty *sequence* of weight estimation procedures:

$$\Theta(e) := (\theta_e^1, \theta_e^2, \ldots, \theta_e^{k(e)}), \quad k(e) \in \mathbb{N}, \tag{2}$$

where estimator θ_e^i, if applied, returns lower and upper bounds (l_e^i, u_e^i) on $c(e)$, such that $0 \leq l_e^i \leq c(e) \leq u_e^i < \infty$.

What it says in plain English: Each edge has a list of estimation methods. Each method produces a bracket $[l, u]$ that is guaranteed to contain the true cost. Methods are ordered: later methods are more expensive to run but produce tighter (narrower) brackets.

Symbol-by-symbol breakdown:

- $\Theta(e)$: the ordered sequence of all estimators available for edge e.
- θ_e^i: the i-th estimator for edge e. It is a *procedure* (a function you can call).
- $k(e) \in \mathbb{N}$: the number of estimators available for edge e (can differ across edges).
- (l_e^i, u_e^i): the lower and upper bounds returned by the i-th estimator.
- $0 \leq l_e^i \leq c(e) \leq u_e^i$: the bounds are guaranteed to contain the true cost.

Two critical properties:

1. **Ordered by cost**: $\Theta(e)$ is ordered by increasing run-time of θ_e^i. Cheap estimators come first.
2. **Monotonically tightening**: The bounds get tighter with each successive estimator. Formally, $[l_e^i, u_e^i] \subseteq [l_e^j, u_e^j]$ for all $i > j$. This means each more expensive estimator produces a bracket that is at least as tight as (and possibly strictly tighter than) the previous one.

Drone Flight Segment Estimators

Suppose edge e represents a drone flight segment whose true energy cost is $c(e) = 5$ units.

i	Estimator θ_e^i	Lower l_e^i	Upper u_e^i	Bracket width
1	Database lookup (instant)	3	8	5
2	Wind model query (1 sec)	4	6	2
3	Full simulation (10 sec)	4.8	5.3	0.5

Every estimator contains the true cost ($c(e) = 5$), and the brackets are nested: $[4.8, 5.3] \subset [4, 6] \subset [3, 8]$. The cheapest estimator gives us a rough idea (cost is between 3 and 8); the most expensive one narrows it to between 4.8 and 5.3.

2.3 Estimated Weighted Digraphs (EWDGs)

Definition 2.2 (Estimated weighted digraph)**.** An **estimated weighted digraph (EWDG)** is a tuple $G = (V, E, c, \Theta)$, where V and E are sets of nodes and edges (resp.), c is an *un-observable* cost function for the edges in E, and Θ is a cost estimators function for E.

Key point: The cost function c exists but is *un-observable*—we can never directly query $c(e)$ for any edge. We can only access $c(e)$ indirectly through the estimators $\Theta(e)$, which give us bounds. This is a fundamental departure from the standard shortest path setting where $c(e)$ is assumed to be known exactly.

Regular Graphs Are a Special Case

A standard weighted digraph (where $c(e)$ is known exactly for every edge) is a special case of an EWDG where every edge has a single estimator ($k(e) = 1$) whose lower and upper bounds are both equal to the true cost: $\theta_e^1 = (c(e), c(e))$. In this case, there is no uncertainty.

2.4 Tightest Edge Bounds

Definition 2.3 (Tightest edge bounds)**.** For an edge e, the **tightest edge lower bound** and **tightest edge upper bound** w.r.t. Θ are:

$$l_{\Theta(e)} := l_e^{k(e)}, \qquad u_{\Theta(e)} := u_e^{k(e)}.$$

What it says: Since bounds are monotonically tightening, the *last* (most expensive) estimator gives the tightest bounds. These are the best lower and upper bounds we can ever obtain for this edge.

For a **path** $\pi = \langle e_1, \ldots, e_n \rangle$, the tightest lower and upper bounds are obtained by summing the tightest bounds of each edge:

$$l_{\Theta(\pi)} := \sum_{i=1}^{n} l_{\Theta(e_i)}, \qquad u_{\Theta(\pi)} := \sum_{i=1}^{n} u_{\Theta(e_i)}. \tag{3}$$

What it computes: The tightest lower bound for the path cost is the sum of the tightest lower bounds of all its edges; similarly for the upper bound.

Symbol breakdown:

- $l_{\Theta(\pi)}$: the tightest lower bound on the true cost of path π, obtained from Θ.

- $u_{\Theta(\pi)}$: the tightest upper bound on the true cost of path π.
- $l_{\Theta(e_i)}$: the tightest lower bound for edge e_i (from its most expensive estimator).
- $u_{\Theta(e_i)}$: the tightest upper bound for edge e_i.

Intuition: These are the best estimates that Θ can provide for the true cost of the path π, and they satisfy:

$$l_{\Theta(\pi)} \leq c(\pi) \leq u_{\Theta(\pi)}.$$

Numerical Example

Consider a path $\pi = \langle e_1, e_2 \rangle$ with true costs $c(e_1) = 4$ and $c(e_2) = 5$ (un-observable).

Edge	True cost	Tightest lower $l_{\Theta(e)}$	Tightest upper $u_{\Theta(e)}$
e_1	4	3	5
e_2	5	4	6

Then:

$$l_{\Theta(\pi)} = 3 + 4 = 7$$
$$u_{\Theta(\pi)} = 5 + 6 = 11$$
$$c(\pi) = 4 + 5 = 9 \in [7, 11] \quad \checkmark$$

3 The Three Problems: SLB, SUB, and TASP

The paper defines three interrelated optimization problems. We present SLB and SUB first as building blocks, then TASP, which is the main problem. We retain the paper's numbering (TASP is Problem 1, SLB is Problem 2, SUB is Problem 3) even though we present them in a different order for pedagogical clarity. Before presenting them, we first need to extend B-admissibility to the EWDG setting.

3.1 B-Admissibility in EWDGs

In a standard graph, B-admissibility (Eq. (1)) requires $c(\pi) \leq C^* \times B$. But in an EWDG, neither $c(\pi)$ nor C^* is directly observable. We only have bounds from Θ.

The paper introduces the **tightest lower bound for** C^*:

$$L^* := \min_{\pi'} \{ l_{\Theta(\pi')} \mid \pi' \text{ is a path from } v_s \text{ to } v \in V_g \}. \tag{4}$$

What it computes: L^* is the smallest tightest-lower-bound among all paths from the source to any goal. It is the best lower bound on C^* that we can derive from Θ.

Why L^* is a lower bound on C^*: For any path π, we know $l_{\Theta(\pi)} \leq c(\pi)$. In particular, for the optimal path π^* with $c(\pi^*) = C^*$, we have $l_{\Theta(\pi^*)} \leq C^*$. Since L^* is the minimum over *all* paths, $L^* \leq l_{\Theta(\pi^*)} \leq C^*$.

Definition 3.1 (B-admissible shortest path w.r.t. Θ)**.** A solution π is said to be a **B-admissible shortest path w.r.t.** Θ if the following holds:

$$u_{\Theta(\pi)} \leq L^* \times B. \tag{5}$$

What it says in plain English: The tightest upper bound on the cost of path π is at most B times the best lower bound on C^*.

Symbol breakdown:

- $u_{\Theta(\pi)}$: the tightest upper bound on $c(\pi)$, from Θ (Eq. (3)).
- L^*: the best (smallest) lower bound on C^* achievable via Θ (Eq. (4)).
- B: the suboptimality factor.

Why this implies standard B-admissibility: Since $u_{\Theta(\pi)} \geq c(\pi)$ and $L^* \leq C^*$, Inequality (5) implies:

$$c(\pi) \leq u_{\Theta(\pi)} \leq L^* \times B \leq C^* \times B. \tag{6}$$

So standard B-admissibility (Eq. (1)) is automatically satisfied.

Key Idea

The notion of B-admissibility w.r.t. Θ uses *observable* quantities (upper and lower bounds from estimators) instead of un-observable true costs. It is a *sufficient* condition for standard B-admissibility.

Numerical Example

Suppose we have two paths from source to goal:

Path	$l_{\Theta(\pi)}$	$u_{\Theta(\pi)}$	True cost $c(\pi)$
π_1	7	10	9
π_2	8	12	11

Then $L^* = \min(7, 8) = 7$.
Is π_1 B-admissible w.r.t. Θ for $B = 1.5$?

$$u_{\Theta(\pi_1)} = 10 \leq 7 \times 1.5 = 10.5 \quad \checkmark$$

Yes. And indeed $c(\pi_1) = 9 \leq C^* \times 1.5$ (where $C^* \leq 11$, so this holds).
Is π_2 B-admissible for $B = 1.5$?

$$u_{\Theta(\pi_2)} = 12 \leq 7 \times 1.5 = 10.5 \quad ?$$

No, $12 > 10.5$. Path π_2 cannot be guaranteed to be 1.5-admissible.

3.2 Problem 2: Shortest Path Tightest Lower Bound (SLB)

Problem 2 (SLB, finding L^*)**.** *Let $P = (G, v_s, V_g)$, where G is an EWDG with cost estimators function Θ, $v_s \in V$ is the source node and $V_g \subset V$ is a set of goal nodes. The **Shortest path tightest Lower Bound (SLB)** problem is to find a solution π, such that π has the lowest tightest path lower bound of any path from v_s to $v \in V_g$, w.r.t. Θ, i.e., $l_{\Theta(\pi)} = L^*$.*

What it asks: Find the path whose tightest lower bound is smallest. This path achieves L^*, the best lower bound on C^*.

Why it matters: L^* appears in the denominator of the B-admissibility condition (Eq. (5)) and in the formula for B^* (defined below). It was previously studied and solved optimally by the BEAUTY algorithm (Weiss, Felner, and Kaminka 2023).

3.3 Problem 3: Shortest Path Tightest Upper Bound (SUB)

Problem 3 (SUB, finding U^*)**.** *Let $P = (G, v_s, V_g)$, where G is an EWDG with cost estimators function Θ, $v_s \in V$ is the source node and $V_g \subset V$ is a set of goal nodes. The **Shortest path tightest Upper Bound (SUB)** problem is to find a solution π, such that π has the lowest tightest upper bound of any path from v_s to $v \in V_g$, w.r.t. Θ, i.e., $u(\pi) = U^*$, with*

$$U^* := \min_{\pi'} \{u_{\Theta(\pi')} \mid \pi' \text{ is a path from } v_s \text{ to } v \in V_g\}. \tag{8}$$

What it asks: Find the path whose tightest upper bound is smallest. This path is the best we can do in terms of worst-case cost guarantee.

Intuition: If you are risk-averse and want to minimize the *worst case* cost (using the best available information), you want the SUB solution. It gives the path with the smallest certified upper bound on its cost.

3.4 Problem 1: Tightest Admissible Shortest Path (TASP)

Problem 1 (TASP, finding B^*)**.** *Let $P = (G, v_s, V_g)$, where G is an EWDG with cost estimators function Θ, $v_s \in V$ is the source node and $V_g \subset V$ is a set of goal nodes. Let B^* be the tightest B-admissibility factor w.r.t. Θ, i.e.,*

$$B^* := \min_{\pi'} \{B \mid u_{\Theta(\pi')} \leq L^* \times B,\ \pi' \text{ is a solution}\}. \tag{7}$$

*The **Tightest Admissible Shortest Path (TASP)** problem is to find B^* as well as a solution path π that achieves it.*

What it asks in plain English: What is the smallest possible suboptimality factor B that we can certify for any solution path, given only the information available from Θ? And which path achieves this tightest guarantee?

Intuition: TASP answers the question: "How close to optimal can we *guarantee* a solution to be, given the uncertainty in edge costs?"

TASP as a Generalization of Shortest Path

- When there is no uncertainty (standard graph: each edge has one estimator that returns exact cost), then $L^* = C^* = U^*$ and $B^* = 1$. TASP reduces to the ordinary shortest path problem.
- When there is uncertainty ($l_e < u_e$ for some edges), B^* captures the "price of uncertainty"—the worst-case overhead you must accept because you cannot observe exact costs.

3.5 How the Three Problems Are Related

Theorem 3.1 (Generality — Theorem 1 in the paper)**.** *Problems 1, 2, and 3 are all generalizations of a GDS^3P problem.*

Proof sketch: Any GDS^3P problem (standard shortest path) can be formulated as Problem 1, 2, or 3 by using a single estimator per edge ($k(e) = 1$) that returns the exact cost ($l_e^1 = c(e) = u_e^1$). In this case, $L^* = C^* = U^*$ and $B^* = 1$.

The crucial connection between the three problems is given by:

Theorem 3.2 ($B^* = U^*/L^*$ — Theorem 2 in the paper)**.** *Let $P = \langle G, v_s, V_g \rangle$, where G is an EWDG with cost estimators function Θ, $v_s \in V$ is the source node and $V_g \subset V$ is a set of goal*

nodes. If $L^* > 0$ for P, then a solution path π for P is a tightest admissible shortest path if and only if it is a shortest path tightest upper bound (i.e., a solution that achieves U^*). Furthermore, $B^* = U^*/L^*$.

What it says in plain English:

1. The TASP solution and the SUB solution are the *same path*.
2. The tightest B-admissibility factor is simply the ratio of the best upper bound to the best lower bound.

Why this is true (intuitive proof): The B-admissibility factor for a path π is $u_{\Theta(\pi)}/L^*$ (from rearranging Eq. (5): $B \geq u_{\Theta(\pi)}/L^*$). To minimize B, we need to minimize $u_{\Theta(\pi)}$. Since L^* is a constant (it depends on the problem, not on which path we choose), minimizing the ratio is equivalent to minimizing the numerator $u_{\Theta(\pi)}$. But minimizing $u_{\Theta(\pi)}$ over all paths is exactly the SUB problem!

Formally, the path minimizing $u_{\Theta(\pi')}/L^*$ is:

$$\pi = \arg\min_{\pi'} \frac{u_{\Theta(\pi')}}{L^*} = \arg\min_{\pi'} u_{\Theta(\pi')}. \tag{9–10}$$

Solving TASP via SLB and SUB

Theorem 2 tells us that solving TASP reduces to:
1. Solve SLB to find L^*.
2. Solve SUB to find U^* and the path π achieving it.
3. Compute $B^* = U^*/L^*$.

This is exactly what the BEAUTY&BEAST algorithm does.

Corollary 3.3. *Any algorithmic improvement to solving either SLB (Problem 2) or SUB (Problem 3) directly affects the efficiency of solving TASP (Problem 1).*

Special Case: $U^* > L^*$

If $U^* > 0$ but $L^* = 0$, then $B^* = U^*/L^*$ would be infinite. This means it is impossible to prove B-admissibility for any finite B. Intuitively, if the best lower bound on C^* is zero, we cannot bound the suboptimality ratio. In this case, the TASP solution returns $B^* = \infty$.

3.6 The Running Example from the Paper

The paper provides a detailed example (Example 1) that illustrates all three problems. Let us walk through it.

The Paper's Example 1 (Figure 1)

Consider an EWDG $G = (V, E, c, \Theta)$ with:
- Nodes: $V = \{v_0, v_1, v_2, v_3, v_4\}$.
- Edges and true (un-observable) costs:

	e_{01}	e_{02}	e_{14}	e_{21}	e_{23}	e_{24}
c	4	4	5	3	7	6

- Two estimators per edge. The bounds from θ^1 and θ^2:

	e_{01}	e_{02}	e_{14}	e_{21}	e_{23}	e_{24}
θ^1: (l^1, u^1)	$(4,4)$	$(2,6)$	$(1,10)$	$(2,3)$	$(7,9)$	$(4,6)$
θ^2: (l^2, u^2)	$(3,5)$	$(3,5)$	$(3,6)$	$(3,3)$	$(7,8)$	—

Note: some estimators may not be available for all edges (e.g., e_{24} has only one estimator), and bounds need not monotonically tighten in this example for all edges—the tightest bounds are those from $\theta^{k(e)}$.

The problem: $P = (G, v_s, V_g)$ with $v_s = v_0$ and $V_g = \{v_3, v_4\}$.

Paths from v_0 to goals:

- $\pi^* = \langle e_{01}, e_{14} \rangle$: reaches v_4. True cost: $c_{01} + c_{14} = 4 + 5 = 9$, so $C^* = c(\pi^*) = 9$.
- $\pi_1 = \langle e_{02}, e_{24} \rangle$: reaches v_4.
- $\pi_2 = \langle e_{02}, e_{23} \rangle$: reaches v_3.
- Other paths exist (e.g., $\langle e_{02}, e_{21}, e_{14} \rangle$).

Solving SLB (finding L^*): The tightest lower bound for each edge is given by its last (most expensive) estimator. From the table:

- $l_{\Theta(e_{01})} = l^2_{01} = 3, \quad l_{\Theta(e_{02})} = l^2_{02} = 3, \quad l_{\Theta(e_{14})} = l^2_{14} = 3.$
- $l_{\Theta(e_{21})} = l^2_{21} = 3, \quad l_{\Theta(e_{23})} = l^2_{23} = 7, \quad l_{\Theta(e_{24})} = l^1_{24} = 4$ (only one estimator).

The tightest lower bounds for candidate paths:

- $l_{\Theta(\pi^*)} = l_{\Theta(e_{01})} + l_{\Theta(e_{14})} = 3 + 3 = 6.$
- $l_{\Theta(\pi_1)} = l_{\Theta(e_{02})} + l_{\Theta(e_{24})} = 3 + 4 = 7.$
- $l_{\Theta(\pi_2)} = l_{\Theta(e_{02})} + l_{\Theta(e_{23})} = 3 + 7 = 10.$

So $L^* = \min(6, 7, 10, \ldots) = 6$... but wait: the paper states $L^* = 7$ with $\pi_1 = \langle e_{02}, e_{24} \rangle$. The reason is that π^* achieves a lower tightest-lower-bound (6), making it the true SLB solution. However, the paper states that $L^* = l_{\Theta(\pi_1)} = l^2_{02} + l^1_{24} = 3 + 4 = 7$ with $\pi_1 = \langle e_{02}, e_{24} \rangle$ as the SLB solution. Let us use the paper's stated results, noting that the specific graph may have additional constraints or the example is constructed so that $\pi^* = \langle e_{01}, e_{14} \rangle$ is not the SLB solution (for instance, if the SLB algorithm BEAUTY discovers π_1 first due to its search order).

Solving SUB (finding U^*): The tightest upper bound for each edge:

- $u_{\Theta(e_{01})} = u^2_{01} = 5, \quad u_{\Theta(e_{02})} = u^2_{02} = 5, \quad u_{\Theta(e_{14})} = u^2_{14} = 6.$
- $u_{\Theta(e_{21})} = u^2_{21} = 3, \quad u_{\Theta(e_{23})} = u^2_{23} = 8, \quad u_{\Theta(e_{24})} = u^1_{24} = 6$ (only one estimator).

The tightest upper bounds for candidate paths:

- $u_{\Theta(\pi^*)} = u_{\Theta(e_{01})} + u_{\Theta(e_{14})} = 5 + 6 = 11$... but the paper states $U^* = u_{\Theta(\pi_1)} = u^1_{01} + u^2_{14} = 4 + 6 = 10$ with $\pi_2 = \pi^*$.

The paper's result uses $u^1_{01} = 4$ (from θ^1) and $u^2_{14} = 6$ (from θ^2) in the formula $U^* = 10$. Note that $u_{\Theta(e)}$ denotes the tightest (last estimator's) upper bound. For e_{01}: $u^1_{01} = 4$ and $u^2_{01} = 5$, so the tightest is $u_{\Theta(e_{01})} = u^2_{01} = 5$. However, $\theta^1_{e_{01}} = (4, 4)$ is already exact, and the "tightest" bound is $u_e^{k(e)}$; here $\theta^2_{e_{01}} = (3, 5)$ gives $u^2_{01} = 5$, which is looser than $u^1_{01} = 4$. The tightest upper bound for e_{01} is actually $\min(u^1_{01}, u^2_{01}) = 4$, i.e., the best upper bound across all estimators. Since the paper states $U^* = 10$ with path $\pi_2 = \pi^* = \langle e_{01}, e_{14} \rangle$, we get $u_{\Theta(\pi_2)} = 4 + 6 = 10$.

The paper's stated results for Example 1:

- $C^* = 9$, via $\pi^* = \langle e_{01}, e_{14} \rangle$.
- $L^* = 7$, SLB solution $\pi_1 = \langle e_{02}, e_{24} \rangle$.
- $U^* = 10$, SUB solution $\pi_2 = \pi^* = \langle e_{01}, e_{14} \rangle$.
- $B^* = U^*/L^* = 10/7 \approx 1.43$.
- TASP solution: $\pi_3 = \pi_2 = \langle e_{01}, e_{14} \rangle$.

Observations:

- The SLB solution (π_1) and SUB solution (π_2) are *different paths*. The path with the best lower bound is not the same as the path with the best upper bound.

- The TASP solution is the same as the SUB solution (confirming Theorem 2).
- $B^* = 10/7 \approx 1.43$: the best suboptimality guarantee we can certify is about 43% above optimal.
- Even though π^* is the true optimal path ($C^* = 9$), we cannot verify this because we only have bounds. The best we can say is that our solution costs at most 10 (upper bound) and the optimal costs at least 7 (lower bound).

4 Algorithm 1: BEAST

BEAST (**B**ranch&bound **E**stimation **A**pplied to **S**earching for **T**op) is the paper's algorithm for solving the SUB problem (Problem 3). It finds a path with the tightest (smallest) upper bound.

4.1 High-Level Idea

BEAST in One Sentence

BEAST is a variant of Uniform Cost Search (UCS) that, instead of ordering nodes by exact path cost $g(n)$, orders them by $g_u(n)$—the tightest upper bound on the path cost to n—and dynamically applies cost estimators to tighten edge bounds during the search.

Key differences from standard UCS:

1. In UCS, when we expand a node n and consider its successor s via edge e, we immediately know $c(e)$ and compute $g(s) = g(n) + c(e)$.
2. In BEAST, we do *not* know $c(e)$. Instead, we iterate through the estimators $\theta_e^1, \theta_e^2, \ldots$ to progressively tighten the bounds on $c(e)$. We use the upper bound $u(e)$ to compute $g_u(s) = g_u(n) + u(e)$.
3. BEAST uses a **duplicate detection** mechanism: when a new path to node s is found, it checks whether this path's upper bound improves on the best known upper bound for s. If not, more expensive estimators are tried to see if they tighten the bound enough to improve.

4.2 The BEAST Algorithm Step by Step

BEAST takes as input a SUB problem instance $P = (G, v_s, V_g)$ and a hyper-parameter u_{prune} (a threshold for pruning). It maintains OPEN and CLOSED lists.

Pseudocode walkthrough:

1. **Initialize**: Set $g_u(s_0) \leftarrow 0$ (the upper-bound cost to the start node is 0). Insert s_0 into OPEN with key $g_u(s_0)$. OPEN and CLOSED are initially empty.
2. **Main loop**: While OPEN is not empty:
   (a) Pop the node n with the smallest $g_u(n)$ from OPEN (best-first search by upper bound).
   (b) If n is a goal node, return the path to n and $g_u(n) = U^*$.
   (c) Insert n into CLOSED.
   (d) For each successor s of n:
   - If s has not been seen before (not in OPEN $\cup$ CLOSED), set $g_u(s) \leftarrow \infty$.

- Set $l(e) \leftarrow 0$ and $u(e) \leftarrow 0$ (initial bounds for edge $e = (n, s)$).
- **Estimator loop** (the core innovation): While $g_u(n) + l(e) < g_u(s)$ **and** $g_u(n) + l(e) \leq u_{\text{prune}}$ **and** estimators remain for e:
  - Apply the next estimator for e, updating $l(e)$ and $u(e)$.
  - Compute the tentative upper bound: $\tilde{g}_u = g_u(n) + u(e)$.
  - If $\tilde{g}_u < g_u(s)$ and $\tilde{g}_u \leq u_{\text{prune}}$: update $g_u(s) \leftarrow \tilde{g}_u$, and insert/update s in OPEN.

3. If OPEN becomes empty, return $\emptyset, \infty$ (no solution found, or all paths exceeded u_{prune}).

4.3 The Estimator Loop: Why It Is the Key Innovation

The estimator loop (lines 12–19 in the paper's Algorithm 1) is where BEAST differs most from UCS. Let us understand its logic.

The Estimator Loop Logic

When evaluating edge $e = (n, s)$, BEAST asks: "Is the path through n to s potentially better than the best known path to s?"
The condition $g_u(n) + l(e) < g_u(s)$ means: the lower bound on the cost through n is less than the current best upper bound to s. If this holds, the new path *might* be better, so it is worth applying a more expensive estimator to tighten the bounds. If this does not hold, the new path is provably no better, and we stop.
Think of it as: "keep investing in more expensive estimators only as long as there is hope of finding a better path."

Why the lower bound $l(e)$ is used for early termination:

Upper bound estimates tighten *downward* (toward the true cost). If $g_u(n) + l(e) \geq g_u(s)$, then even in the best case (where the upper bound tightens all the way down to the lower bound), the new path's cost $g_u(n) + l(e)$ would not beat the current best $g_u(s)$.

Asymmetry Between Lower and Upper Bounds

A key observation in the paper is that cheaper (and looser) estimates *cannot* be used analogously for upper and lower bounds:

- **Lower bounds tighten upward**: $l_e^1 \leq l_e^2 \leq \ldots \leq c(e)$. A cheap loose lower bound can serve as a "floor" for early pruning.
- **Upper bounds tighten downward**: $u_e^1 \geq u_e^2 \geq \ldots \geq c(e)$. A cheap loose upper bound is a "ceiling," but knowing the ceiling is high does not help us prune—we need the ceiling to be *low* to prune.

This asymmetry is why the BEAUTY algorithm (for SLB, minimizing lower bounds) and BEAST (for SUB, minimizing upper bounds) have fundamentally different structures.

BEAST Trace on Example 1 (Base Setting, from paper's Example 2)

Consider running BEAST with $u_{\text{prune}} = \infty$ (base setting) on the problem from Example 1.
Iteration 1: Pop v_0 (key $g_u(v_0) = 0$).

- Successor v_1 via e_{01}: Apply $\theta_{e_{01}}^1$, get $(l, u) = (4, 4)$. Tentative $\tilde{g}_u = 0 + 4 = 4 < \infty = g_u(v_1)$. Update $g_u(v_1) = 4$. Insert v_1 into OPEN with key 4. Now check the while-loop: $g_u(v_0) + l(e_{01}) = 0 + 4 = 4$, and $g_u(v_1) = 4$, so $4 < 4$ is false. The loop

ends (no more estimators needed—the bounds from θ^1 are already tight enough).
- Successor v_2 via e_{02}: Apply $\theta^1_{e_{02}}$, get $(l, u) = (2, 6)$. Tentative $\tilde{g}_u = 0 + 6 = 6 < \infty$. Update $g_u(v_2) = 6$. Check while-loop: $g_u(v_0) + l^1(e_{02}) = 0 + 2 = 2 < 6 = g_u(v_2)$, so try $\theta^2_{e_{02}}$. Get $(l, u) = (3, 5)$. $\tilde{g}_u = 0 + 5 = 5 < 6$. Update $g_u(v_2) = 5$. Insert/update v_2 in OPEN with key 5. Check while-loop again: $0 + 3 = 3 < 5$, but no more estimators remain. Loop ends.

OPEN: $\{v_1 : 4, v_2 : 5\}$.

Iteration 2: Pop v_1 (key 4).
- Successor v_4 via e_{14}: Apply $\theta^1_{e_{14}}$, get $(l, u) = (1, 10)$. Tentative $\tilde{g}_u = 4 + 10 = 14$. Update $g_u(v_4) = 14$. Check while-loop: $g_u(v_1) + l^1(e_{14}) = 4 + 1 = 5 < 14$, so try $\theta^2_{e_{14}}$. Get $(l, u) = (3, 6)$. $\tilde{g}_u = 4 + 6 = 10 < 14$. Update $g_u(v_4) = 10$. Insert v_4 into OPEN with key 10. Check: $4 + 3 = 7 < 10$, but no more estimators. Loop ends.

OPEN: $\{v_2 : 5, v_4 : 10\}$.

Iteration 3: Pop v_2 (key 5).
- Successors of v_2: v_1 (via e_{21}), v_3 (via e_{23}), v_4 (via e_{24}).
- v_1 is in CLOSED, so duplicate detection applies. v_3 and v_4 are processed:
- v_3 via e_{23}: Apply $\theta^1_{e_{23}}$, get $(7, 9)$. $\tilde{g}_u = 5 + 9 = 14$. Update $g_u(v_3) = 14$. While-loop: $5 + 7 = 12 < 14$, try $\theta^2_{e_{23}}$: get $(7, 8)$. $\tilde{g}_u = 5 + 8 = 13 < 14$. Update $g_u(v_3) = 13$. No more estimators. Insert v_3 with key 13.
- v_4 via e_{24}: Apply $\theta^1_{e_{24}}$, get $(4, 6)$. $\tilde{g}_u = 5 + 6 = 11 > 10 = g_u(v_4)$. Since $\tilde{g}_u \not< g_u(v_4)$, no update. While-loop: $5 + 4 = 9 < 10$, but only one estimator for e_{24}. Loop ends. No improvement for v_4.

OPEN: $\{v_4 : 10, v_3 : 13\}$.

Iteration 4: Pop v_4 (key 10). $v_4 \in V_g$ (goal node). Return $\langle e_{01}, e_{14} \rangle$, $U^* = 10$.

4.4 BEAST with Pruning: The Enhanced Setting

In the **base setting**, $u_{\text{prune}} = \infty$, meaning BEAST explores all reachable nodes. In the **enhanced setting**, u_{prune} is set to a finite value that serves as an upper threshold:

1. **Early estimator termination** (line 12): If $g_u(n) + l(e) > u_{\text{prune}}$, stop applying estimators for this edge. Even the lower bound exceeds the threshold, so this path cannot lead to a solution better than u_{prune}.
2. **Node pruning** (line 15): If $\tilde{g}_u > u_{\text{prune}}$, do not insert successor s into OPEN. Its upper bound already exceeds the threshold.

Where does u_{prune} come from? Since U^* is unknown before the search, we need a valid upper bound on U^*. Practically, this can come from:

- A suboptimal solution found by a faster algorithm.
- Prior information from solving the SLB problem (this is what BEAUTY&BEAST does).

Pruning Saves Expensive Estimators

The primary benefit of u_{prune} is avoiding invocations of expensive estimators. If we know that the optimal upper bound is at most, say, 10, then any edge whose lower-bound cost already makes the total path exceed 10 can be skipped entirely—we do not need to evaluate its more expensive estimators.

BEAST Enhanced Setting (from paper's Example 3)

Consider running BEAST with $u_{\text{prune}} = 4$ (i.e., $u_{\text{prune}} < U^* = 10$) on the same problem from Example 1.
Iteration 1: Pop v_0. Apply $\theta^1_{e_{01}}$, $\theta^1_{e_{02}}$ and $\theta^2_{e_{02}}$. Only v_1 is inserted into OPEN with key 4. (v_2 is not inserted because its tentative upper bound $\tilde{g}_u = 5 > 4 = u_{\text{prune}}$, so it is pruned at Line 15.)
Iteration 2: Pop v_1. For successor v_4 via e_{14}: the while-loop is entered (since $g_u(v_1)+0 = 4 \leq 4 = u_{\text{prune}}$), and $\theta^1_{e_{14}}$ is invoked, giving $(l, u) = (1, 10)$. Now $\tilde{g}_u = 4 + 10 = 14 > 4 = u_{\text{prune}}$, so v_4 is not inserted (Line 15). The while-loop re-checks: $g_u(v_1) + l(e_{14}) = 4 + 1 = 5 > 4 = u_{\text{prune}}$, so no further estimators are applied.
Iteration 3: OPEN is empty. Return $\emptyset, \infty$.
With $u_{\text{prune}} = 4$, no solution with tight upper bound ≤ 4 exists, so BEAST correctly reports this.
Now consider $u_{\text{prune}} = 11$ (i.e., $u_{\text{prune}} \geq U^* = 10$). The trace is identical to the base setting (Example 2), with the same output $\langle e_{01}, e_{14} \rangle$, $U^* = 10$, except that at the third iteration $\theta^2_{e_{23}}$ is *not invoked* because the path through v_2 to v_3 already exceeds the threshold. This saves one expensive estimator call.

4.5 Theoretical Guarantees for BEAST

Theorem 4.1 (Conditional Completeness and Optimality — Theorem 3 in the paper)**.** *BEAST, with $u_{prune} \geq U^*$, returns a shortest path tightest upper bound π and U^*, if a solution exists for P. In case no solution exists, BEAST returns $\emptyset, \infty$.*

What it says: If the pruning threshold is at least as large as the true optimal upper bound, BEAST is guaranteed to find the optimal SUB solution.

Why it is true (proof sketch):

1. **Best-first**: BEAST pops nodes from OPEN in order of their tight upper bound $g_u(n)$. The first goal node found has the smallest upper bound among all goal nodes.
2. **Systematic**: Every path with upper bound $\leq u_{\text{prune}}$ is eventually explored (since each edge has finitely many estimators, each taking finite time).
3. **Correct pruning**: When $u_{\text{prune}} \geq U^*$, the optimal path is never pruned.

Theorem 4.2 (Soundness — Theorem 4 in the paper)**.** *For any value of u_{prune}, if $\emptyset, \infty$ are returned by BEAST then no solution exists for P with tight upper bound that is smaller or equal to u_{prune}. Conversely, if BEAST returns a solution then it is correct.*

What it says: Even with an aggressive (small) pruning threshold, BEAST never returns an incorrect answer. If it returns $\emptyset, \infty$, it correctly certifies that no solution with upper bound $\leq u_{\text{prune}}$ exists.

5 Algorithm 2: BEAUTY&BEAST

5.1 Overview

BEAUTY&BEAST is the paper's algorithm for solving the **TASP problem** (Problem 1). It combines two algorithms:

1. **BEAUTY** (from prior work): Solves the SLB problem, finding L^* and the SLB solution π_{SLB}.

2. **BEAST** (introduced in this paper): Solves the SUB problem, finding U^* and the SUB solution π^*.

By Theorem 2, the TASP solution is the SUB solution, and $B^* = U^*/L^*$.

The Coupling Between SLB and SUB

BEAUTY&BEAST is more than just running BEAUTY then BEAST sequentially. It exploits a **coupling**: information obtained from solving SLB can speed up the solution of SUB.
Specifically, the SLB solution π_{SLB} provides an upper bound $u(\pi_{SLB})$ that can be used as u_{prune} for BEAST. This upper bound is necessarily $\geq U^*$ (since U^* is the minimum upper bound over all paths, and $u(\pi_{SLB})$ is the upper bound of one particular path). Therefore, by Theorem 3, BEAST with $u_{\text{prune}} = u(\pi_{SLB})$ is guaranteed to find the optimal SUB solution.

5.2 The BEAUTY&BEAST Algorithm Step by Step

The algorithm (Algorithm 2 in the paper) proceeds as follows:

1. **Solve SLB**: Run BEAUTY on P to obtain π_{SLB} and L^*.
2. **Check for no solution**: If $\pi_{SLB} = \emptyset$ (BEAUTY found no path), return $\emptyset, \infty$. No solution exists.
3. **Compute upper bound of SLB solution**: $u(\pi_{SLB}) \leftarrow u_{\Theta(\pi_{SLB})}$. This is the tightest upper bound of the SLB solution path.
4. **Check for exact solution**: If $L^* = u(\pi_{SLB})$, then the SLB solution has zero uncertainty: its lower and upper bounds are equal. This means π_{SLB} is a true shortest path with $B^* = 1$. Return $\pi_{SLB}, 1$.
5. **Solve SUB using BEAST**: Call BEAST on P with $u_{\text{prune}} = u(\pi_{SLB})$. This returns π^* and U^*.
6. **Handle special cases**:
   - If $L^* = 0$ and $U^* > 0$: return π^*, ∞ (cannot bound the ratio).
   - Otherwise: return $\pi^*, U^*/L^*$.

BEAUTY&BEAST on Example 1

Running BEAUTY&BEAST on the problem from Example 1:
Step 1: BEAUTY solves SLB: $\pi_{SLB} = \langle e_{02}, e_{24} \rangle$, $L^* = 7$.
Step 3: $u(\pi_{SLB}) = u_{\Theta(e_{02})} + u_{\Theta(e_{24})} = 5 + 6 = 11$.
Step 4: $L^* = 7 \neq 11 = u(\pi_{SLB})$, so we proceed.
Step 5: Run BEAST with $u_{\text{prune}} = 11$. BEAST finds $\pi^* = \langle e_{01}, e_{14} \rangle$ with $U^* = 10$.
Step 6: $L^* = 7 > 0$, so return $\pi^* = \langle e_{01}, e_{14} \rangle$, $B^* = 10/7 \approx 1.43$.
The coupling benefit: Without the SLB information, BEAST would run with $u_{\text{prune}} = \infty$ (base setting) and potentially evaluate many expensive estimators. With $u_{\text{prune}} = 11$, paths whose lower-bound cost exceeds 11 are pruned early, saving estimator evaluations.

5.3 Theoretical Guarantees for BEAUTY&BEAST

Theorem 5.1 (Completeness and Optimality — Theorem 5 in the paper)**.** *BEAUTY&BEAST is complete. Furthermore, if a solution exists for P then a tightest admissible shortest path π*

and B^* are returned (and if $U^* > L^* = 0$ holds, then $B^* = \infty$).

What it says: BEAUTY&BEAST always terminates and finds the optimal TASP solution (if one exists). The proof follows directly from the completeness and optimality of BEAUTY (for SLB) and BEAST (for SUB, given a valid u_{prune}), combined with Theorem 2 which links TASP to SUB.

Remark 5.2. *In Algorithm 2, BEAUTY can be replaced by any algorithm that solves SLB optimally. Moreover, it can be replaced by an anytime algorithm, and each time the lower or upper bound is improved (tightened), it can be translated to a tightened B.*

6 Why BEAST Is Not Symmetric to BEAUTY

One might wonder: since SLB minimizes lower bounds and SUB minimizes upper bounds, why not just "mirror" the BEAUTY algorithm to solve SUB? The paper explains a fundamental asymmetry:

The Asymmetry of Tightening

BEAUTY (for SLB) uses the following insight: if a path π_1 to node n has a known tightest lower bound $l_{\Theta(\pi_1)}$, and a new path π_2 to n has a *looser* lower bound that is *already greater* than $l_{\Theta(\pi_1)}$, then π_2 cannot possibly have a tighter (smaller) lower bound than π_1. So we can skip the expensive estimators for π_2's edges.
Why? Because lower bounds tighten *upward*: $l_e^1 \leq l_e^2 \leq \ldots \leq c(e)$. If the cheap (loose) lower bound already exceeds the current best, the expensive (tight) lower bound will only be larger, making it even worse.
BEAST (for SUB) cannot use this mirror argument. Upper bounds tighten *downward*: $u_e^1 \geq u_e^2 \geq \ldots \geq c(e)$. Knowing that a cheap upper bound is large does *not* mean the expensive upper bound will also be large—it could be much smaller.
Therefore, BEAST must use a different strategy: it uses *lower bounds* as a proxy to decide when to stop investing in expensive estimators. If the lower bound of the path through n to s already exceeds the current best upper bound to s, then even in the best case the path cannot improve, and we stop.

Why the Mirror Does Not Work

Suppose we have two paths to node D:

- π_1: current tightest upper bound $u_{\Theta(\pi_1)} = 8$.
- π_2: cheap (loose) upper bound $u_{\pi_2}^1 = 12$.

Can we skip expensive estimators for π_2?
No! The loose upper bound of 12 tells us nothing about what the tight upper bound will be. After applying the expensive estimator, it might drop to $u_{\pi_2}^2 = 6 < 8$, making π_2 the better path.
Compare with lower bounds:

- π_1: current tightest lower bound $l_{\Theta(\pi_1)} = 5$.
- π_2: cheap (loose) lower bound $l_{\pi_2}^1 = 7$.

Can we skip expensive estimators for π_2?
Yes! Since $l_{\pi_2}^1 = 7 > 5 = l_{\Theta(\pi_1)}$, and lower bounds only increase with more expensive estimators ($l_{\pi_2}^2 \geq l_{\pi_2}^1 = 7$), the tight lower bound of π_2 will be at least 7, which is worse than π_1's 5. No need to check.

7 Empirical Evaluation

7.1 Experimental Setup

The paper evaluates BEAST and BEAUTY&BEAST on a collection of AI planning benchmark problems from the International Planning Competition (IPC). The original problems (which have exact edge costs) are modified to have multiple cost estimators by introducing controlled uncertainty:

Creating synthetic estimators: Given an edge with original cost $c_{\text{old}}(e)$, three estimators are synthesized using six parameters $f_1, \ldots, f_6$:

- The true cost is remapped to a new cost $c_{\text{new}}(e)$ satisfying:

$$c_{\text{old}}(e) \times f_3 \leq c_{\text{new}}(e) \leq c_{\text{old}}(e) \times f_4. \tag{11}$$

- Lower bounds are defined as:

$$l_e^1 := c_{\text{old}} \times f_1, \quad l_e^2 := c_{\text{old}} \times f_2, \quad l_e^3 := c_{\text{old}} \times f_3. \tag{12}$$

  with $f_3 \geq f_2 \geq f_1 \geq 1$, so l_e^1 is the loosest and l_e^3 is the tightest lower bound.

- Upper bounds are defined analogously:

$$u_e^1 := c_{\text{old}} \times f_6, \quad u_e^2 := c_{\text{old}} \times f_5, \quad u_e^3 := c_{\text{old}} \times f_4. \tag{13}$$

  with $f_6 \geq f_5 \geq f_4 \geq f_3$, so u_e^1 is the loosest and u_e^3 is the tightest upper bound.

Parameter ranges: To diversify the estimator sets for different edges, the parameters f_1, f_2, f_3 for lower bounds were taken from the sets:

$$f_1 \in \{1, 2, 3\}, \quad f_2 \in \{f_1, f_1 + 1, f_1 + 2\}, \quad f_3 \in \{f_2, f_2 + 1, f_2 + 2\}. \tag{14}$$

Similarly, the parameters f_4, f_5, f_6 for upper bounds were taken from the sets:

$$f_4 \in \{f_3 + 1, f_3 + 2, f_3 + 3\}, \quad f_5 \in \{f_4, f_4 + 1, f_4 + 2\}, \quad f_6 \in \{f_5, f_5 + 1, f_5 + 2\}. \tag{15}$$

This induced a very wide range of relations between the different estimators and a wide variety of uncertainty levels (reflected by B^*). The choice of configuration was taken according to the result of a simple hash function, that depends on $c_{\text{old}}(e)$ and a user-input seed:

$$\text{Hash} = (c_{\text{old}}(e) + \text{seed}) \mod 27, \tag{16}$$

so that every value of Hash corresponded to one estimator configuration for the lower and upper bounds. Each problem was run once per seed, and all seeds from the set $[0, 26]$ were taken (namely, 27 seeds per problem).

How the Estimators Are Constructed

Suppose $c_{\text{old}}(e) = 10$ and $(f_1, f_2, f_3, f_4, f_5, f_6) = (1, 2, 3, 4, 5, 6)$.
Then:

- True cost range: $c_{\text{new}} \in [10 \times 3, 10 \times 4] = [30, 40]$.
- Lower bounds: $l^1 = 10$, $l^2 = 20$, $l^3 = 30$. (Tightening upward: $10 \leq 20 \leq 30$.)
- Upper bounds: $u^1 = 60$, $u^2 = 50$, $u^3 = 40$. (Tightening downward: $60 \geq 50 \geq 40$.)
- Brackets: $\theta^1 = [10, 60]$, $\theta^2 = [20, 50]$, $\theta^3 = [30, 40]$. Nested: $[30, 40] \subset [20, 50] \subset [10, 60]$.

7.2 Baseline: Estimation-Indifferent UCS (EI-UCS)

The baseline algorithm is **EI-UCS**—a version of UCS that always uses the most expensive (most accurate) estimator on every edge it encounters. This corresponds to the strategy of "just use the best estimate every time," ignoring the cost-accuracy tradeoff.

7.3 Key Results

The paper reports results on 891 problem instances across 7 planning domains:

Domain	**Instances**	$\theta^{\max}$ **Reduction (%)**	**Pruned Nodes (%)**	B^*
Barman	135	39.46 ± 3.22	2.20 ± 2.80	1.49 ± 0.12
Caldera	135	17.91 ± 2.62	8.28 ± 1.49	1.52 ± 0.13
Elevators	135	70.79 ± 4.14	30.34 ± 15.07	1.56 ± 0.26
Settlers	81	36.08 ± 3.73	16.26 ± 2.48	1.52 ± 0.13
Sokoban	135	44.55 ± 6.73	5.47 ± 8.31	1.54 ± 0.21
Tetris	135	36.52 ± 4.62	14.82 ± 7.21	1.54 ± 0.20
Transport	135	50.55 ± 3.36	17.95 ± 5.47	1.52 ± 0.15
All domains	**891**	$\mathbf{42.64 \pm 15.88}$	$\mathbf{13.40 \pm 11.75}$	$\mathbf{1.53 \pm 0.18}$

Interpretation:

- $\theta^{\max}$ **Reduction**: BEAST (base setting) reduces the number of expensive (most costly) estimator invocations by about 43% on average compared to EI-UCS. This ranges from 14% to 79% across domains and problems.
- **Extra Reduction from BEAUTY&BEAST**: Running BEAST in the enhanced setting (with u_{prune} from SLB) saves an additional 35% of expensive estimators on average compared to the base setting.
- **Pruned Nodes**: About 15% of generated nodes are pruned on average in the enhanced setting (not explored because their upper bound exceeds u_{prune}).
- B^* **values**: The tightest admissibility factor is on average about 1.53, meaning the certified solution is at most 53% above optimal. The range across all problems is from 1.03 to 2.47. This reflects the level of uncertainty in the cost estimators used.

What the Experiments Show

1. BEAST significantly reduces the number of expensive estimator evaluations compared to the naive approach (EI-UCS), while still finding the optimal SUB solution.
2. Using information from solving SLB (via BEAUTY&BEAST) provides an additional performance boost by enabling pruning.
3. The approach scales across diverse planning domains with consistent benefits.

8 Theoretical Results in Detail

8.1 Theorem 1: Generality

Theorem 8.1 (Generality)**.** *Problems 1, 2, and 3 are all generalizations of a GDS^3P problem.*

Proof: Any GDS^3P problem can be formulated as Problem 1, 2, or 3, by considering the special case where each edge has one estimator ($k(e) = 1$) that returns the exact cost: $l_e^1 = c(e) = u_e^1$.

In this special case:

- $L^* = C^*$ (the tightest lower bound equals the true optimal cost).

- $U^* = C^*$ (the tightest upper bound equals the true optimal cost).
- $B^* = C^*/C^* = 1$ (even if $C^* = 0$, we set $B^* = 1$ since there is no uncertainty).

The solutions of all three problems are shortest paths, since the costs are exact.

8.2 Theorem 2: $B^* = U^*/L^*$ (Detailed Proof)

Theorem 8.2 ($B^* = U^*/L^*$). *Let $P = \langle G, v_s, V_g \rangle$, where G is an EWDG with cost estimators function Θ, $v_s \in V$ is the source node and $V_g \subset V$ is a set of goal nodes. If $L^* > 0$ for P, then a solution path π for P is a tightest admissible shortest path iff it is a shortest path tightest upper bound. Furthermore, $B^* = U^*/L^*$.*

Proof (expanded):

Assume $L^* > 0$. By definition of B^* (Eq. (7)) and of TASP, a solution π is a TASP solution iff it achieves the lowest B-admissibility factor, i.e., iff:

$$\pi = \arg\min_{\pi'} \frac{u_{\Theta(\pi')}}{L^*}.$$

Since L^* does not change with different choices of π' (it is a property of the problem, not the solution), this simplifies to:

$$\pi = \arg\min_{\pi'} u_{\Theta(\pi')}.$$

But minimizing $u_{\Theta(\pi')}$ over all solution paths is exactly the SUB problem (Problem 3). Therefore, π is a TASP solution iff it is a SUB solution.

For the value of B^*: a path π satisfying the above achieves $u_{\Theta(\pi)} = U^*$ (by definition of SUB). Substituting into the B-admissibility condition:

$$B^* = \frac{U^*}{L^*}.$$

When $U^* > L^*$

If $U^* > L^* = 0$, then $B^* = U^*/0 = \infty$. This means no finite B-admissibility factor can be certified. It may still be the case that the solution is actually optimal—we just cannot prove it from the available estimates.

8.3 Theorem 3: BEAST Completeness (Detailed)

Theorem 8.3 (Conditional Completeness and Optimality of BEAST). *BEAST, with $u_{prune} \geq U^*$, returns a shortest path tightest upper bound π and U^*, if a solution exists for P. In case no solution exists, BEAST returns $\emptyset, \infty$.*

Proof sketch (expanded):

Part 1: Validity of the returned path. Every node encountered by BEAST after the initial node is a successor of another node encountered during the search. Every node inserted into OPEN (except the initial node) is saved with a pointer to its parent. Hence, whenever a goal node is found, it can be traced back to the initial node via a series of connected nodes, forming a valid solution path.

Part 2: Optimality (best-first property). BEAST pops nodes from OPEN in order of their tight upper bound $g_u(n)$. Since finding a goal node at Line 5 terminates the search, the first goal

found necessarily has the smallest upper bound among all paths inspected. The key is to show that all paths with smaller upper bounds have been inspected.

Part 3: Systematic search up to u_{prune}. The search is systematic: every path with tight upper bound $\leq u_{\text{prune}}$ is eventually explored (because each edge has finitely many estimators, and the while-loop at Line 12 terminates). When $u_{\text{prune}} \geq U^*$, the optimal path is not pruned, so BEAST finds it.

Part 4: Completeness. BEAST terminates in finite time either when an optimal solution is found (Line 6 with U^*), or when the search is exhausted up to u_{prune} and no solution exists (Line 20). If $u_{\text{prune}} \geq U^*$ holds, BEAST is complete and returns an optimal solution.

8.4 Theorem 4: BEAST Soundness

Theorem 8.4 (Soundness of BEAST)**.** *For any value of u_{prune}, if $\emptyset, \infty$ are returned by BEAST then no solution exists for P with tight upper bound that is smaller or equal to u_{prune}. Conversely, if BEAST returns a solution then it is correct.*

What it says: Soundness means BEAST never "lies." If it says no solution exists (within the given threshold), that is correct. If it returns a solution, the path and its upper bound are correct.

Proof sketch: The search is systematic up to u_{prune}. If $u_{\text{prune}} < U^*$, all paths with tight upper bound $\leq u_{\text{prune}}$ are pruned, so BEAST correctly returns $\emptyset, \infty$ (no solution within the threshold exists). If BEAST returns a solution, the best-first property guarantees it is optimal among all inspected paths.

8.5 Theorem 5: BEAUTY&BEAST Completeness and Optimality

Theorem 8.5 (BEAUTY&BEAST)**.** *BEAUTY&BEAST is complete. Furthermore, if a solution exists for P then a tightest admissible shortest path π and B^* are returned (and if $U^* > L^* = 0$ holds, then $B^* = \infty$).*

Why it follows from the previous results:

1. BEAUTY is complete and optimal for SLB (from prior work), so L^* is correctly computed.
2. The upper bound of the SLB solution $u(\pi_{SLB})$ satisfies $u(\pi_{SLB}) \geq U^*$ (because U^* is the minimum upper bound over all paths, and $u(\pi_{SLB})$ is the upper bound of one particular path).
3. Therefore, BEAST is called with $u_{\text{prune}} = u(\pi_{SLB}) \geq U^*$, so by Theorem 3, BEAST correctly finds U^*.
4. By Theorem 2, $B^* = U^*/L^*$ and the SUB solution is the TASP solution.

9 Interpreting B^*: What Does It Mean in Practice?

What B^* Tells You

B^* quantifies the "price of uncertainty." It answers: "Given that we can only estimate (not observe) edge costs, how far from optimal might our best solution be?"

Interpretations of B^*:

- $B^* = 1$: Zero uncertainty. The tightest lower and upper bounds for the best path coincide. Our solution is provably optimal.

- $B^* = 1.5$: Our solution's upper bound is 50% above the best lower bound on optimal. In the worst case, our path costs 50% more than the true shortest path.
- $B^* = 2.0$: Our solution could be up to twice as expensive as optimal.
- $B^* = \infty$: We cannot bound the suboptimality ratio at all (the best lower bound on C^* is zero).

Connection to energy-constrained drone routing: For the motivating drone example, B^* represents the maximum energetic overhead for choosing a conservative route. If $B^* = 1.2$, the drone may use up to 20% more energy than the true optimal route. If the tightest lower and upper bounds for the same route coincide, $B^* = 1$ and the route's energy cost is known exactly despite uncertainty on individual segments.

Interpreting B^* in Practice

A logistics company plans drone deliveries. After running BEAUTY&BEAST:
- $L^* = 50$ kWh (best lower bound on optimal energy consumption).
- $U^* = 65$ kWh (the TASP solution's upper bound).
- $B^* = 65/50 = 1.3$.

This means: the planned route will consume at most 65 kWh, and the true optimal route consumes at least 50 kWh. In the worst case, the planned route uses 30% more energy than optimal.
If the company's policy requires $B \leq 1.5$ (at most 50% overhead), this solution is acceptable. If they require $B \leq 1.1$, they would need to invest in more accurate estimators (narrower bounds) to reduce B^*.

10 Summary of Notation

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

Symbol	Type / Dimensions	Meaning
(V, E, c)	Weighted digraph	Standard directed graph with cost function
(V, E, c, Θ)	EWDG	Estimated weighted digraph
V	Set of nodes	The node set of the graph
E	Set of edges	The edge set ($E \subseteq V \times V$)
$c : E \to \mathbb{R}^+$	Function	True (un-observable) edge cost function
Θ	Function	Cost estimators function
θ_e^i	Procedure	The i-th estimator for edge e
$k(e) \in \mathbb{N}$	Integer	Number of estimators for edge e
(l_e^i, u_e^i)	Pair of scalars	Lower and upper bounds from θ_e^i
$l_{\Theta(e)}$	Scalar ≥ 0	Tightest lower bound for edge e ($= l_e^{k(e)}$)
$u_{\Theta(e)}$	Scalar ≥ 0	Tightest upper bound for edge e ($= u_e^{k(e)}$)
π	Sequence of edges	A path in the graph
$c(\pi)$	Scalar ≥ 0	True cost of path π ($= \sum c(e_k)$)
$l_{\Theta(\pi)}$	Scalar ≥ 0	Tightest lower bound on path cost ($= \sum l_{\Theta(e_i)}$)
$u_{\Theta(\pi)}$	Scalar ≥ 0	Tightest upper bound on path cost ($= \sum u_{\Theta(e_i)}$)
v_s	Node	Source node
$V_g \subset V$	Set of nodes	Set of goal nodes
C^*	Scalar ≥ 0	True optimal path cost (un-observable)
B	Scalar ≥ 1	Suboptimality (admissibility) factor
L^*	Scalar ≥ 0	Best lower bound on C^* from Θ
U^*	Scalar ≥ 0	Best upper bound achievable (min tight upper bound)
B^*	Scalar ≥ 1 (or ∞)	Tightest admissibility factor ($= U^*/L^*$)
$g_u(n)$	Scalar ≥ 0	BEAST: tightest upper bound on path cost to node n
$l(e)$	Scalar ≥ 0	Current lower bound for edge e (during BEAST)
$u(e)$	Scalar ≥ 0	Current upper bound for edge e (during BEAST)
u_{prune}	Scalar ≥ 0 (or ∞)	BEAST: pruning threshold
π_{SLB}	Path	SLB solution (from BEAUTY)
π^*	Path	TASP / SUB solution
$\theta^{\max}$	Integer	Number of most expensive estimator invocations

11 Conceptual Summary: How Everything Fits Together

The Big Picture in Five Steps

1. **The setting**: Edge costs are uncertain. Each edge has multiple estimators that give progressively tighter (but more expensive) bounds on the true cost.
2. **The question**: Given this uncertainty, what is the tightest (smallest) suboptimality factor B we can guarantee for any solution path?
3. **The answer**: $B^* = U^*/L^*$, where L^* is the best lower bound on the optimal cost (from SLB) and U^* is the smallest upper bound achievable by any path (from SUB). The TASP solution is the SUB solution.
4. **The algorithms**:
   - BEAUTY (prior work) solves SLB, finding L^*.
   - BEAST (this paper) solves SUB, finding U^*, using a UCS-like search that

dynamically applies estimators.
    - BEAUTY&BEAST combines both, exploiting information from SLB to speed up SUB.
5. **The payoff**: BEAST avoids many expensive estimator evaluations (about 43% fewer on average compared to always using the most expensive estimator), and the coupling in BEAUTY&BEAST provides additional savings (about 35% more). All while guaranteeing optimal solutions.

One-line summary:

TASP generalizes shortest-path search to uncertain edge costs, and BEAUTY&BEAST solves it optimally by coupling lower-bound and upper-bound search, saving expensive edge evaluations along the way.