Technical Companion to

Planning with Multiple Action-Cost Estimates

A Self-Contained Guide for CS Undergraduates

Companion to the paper by Eyal Weiss and Gal A. Kaminka (ICAPS 2023)

Abstract

This document explains every equation, definition, theorem, and algorithm in the paper "Planning with Multiple Action-Cost Estimates" (ICAPS 2023) in detail, assuming only undergraduate-level knowledge of AI planning (STRIPS, action costs, heuristic search with A^*) and basic discrete mathematics. No prior knowledge of planning under cost uncertainty or multiple cost estimators is required. Each formal element is accompanied by (1) a plain-English description of what it says or computes, (2) a symbol-by-symbol breakdown, (3) an intuitive explanation of why the concept makes sense, and (4) concrete numerical examples where helpful.

Contents

1 Background: Classical Planning and Action Costs

This section reviews the standard planning concepts that the paper builds upon. If you are comfortable with STRIPS-style planning, weighted graphs, and A^* search, you may skim this section and proceed to Section 2.

1.1 What Is a Planning Domain?

A **planning domain** describes a world in which an agent can act. Formally, a planning domain is a 4-tuple:

$$\Sigma = (S, A, \gamma, c)$$

Symbol-by-symbol breakdown:

- S — a finite set of **states**. Each state is a complete description of the world at one instant (e.g., the locations of all trucks and packages in a logistics problem).
- A — a finite set of **(ground) actions**. These are the things the agent can do (e.g., "drive truck 1 from city A to city B").
- $\gamma : S \times A \to S$ — the **state-transition function**. A partial function: $\gamma(s, a) = s'$ means that applying action a in state s produces the successor state s'. If $\gamma(s, a)$ is undefined, action a is *not applicable* in state s.
- $c : S \times A \to \mathbb{R}^+$ — the **cost function**. $c(s, a)$ is the non-negative cost of applying action a in state s. The paper simplifies this to $c(a)$ (cost depends only on the action, not the state), which is standard practice.

Numerical Example

Consider a tiny logistics world with 2 cities (A, B) and 1 truck:

- States: $S = \{\text{truck-at-A}, \text{truck-at-B}\}$.
- Actions: $A = \{\text{drive-A-to-B}, \text{drive-B-to-A}\}$.
- Transitions: $\gamma(\text{truck-at-A}, \text{drive-A-to-B}) = \text{truck-at-B}$, etc.
- Costs: $c(\text{drive-A-to-B}) = 120$ (minutes), $c(\text{drive-B-to-A}) = 130$ (minutes).

1.2 Classical Planning Problems, Solutions, and Optimal Cost

Definition 1.1 (Classical Planning Problem)**.** A **classical planning problem** is a triple $\mathcal{P} = (\Sigma, s_0, S_g)$, where:

- Σ is a planning domain,
- $s_0 \in S$ is the **initial state**,
- $S_g \subset S$ is a set of **goal states**.

A **solution** (or **plan**) for $\mathcal{P}$ is a finite sequence of actions $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ such that:

1. a_1 is applicable in s_0,
2. each subsequent action a_i is applicable in the state reached by applying $a_1, \ldots, a_{i-1}$ from s_0,
3. the final state s_n reached after all actions is a goal state: $s_n \in S_g$.

The **plan length** is $|\pi| = n$ (the number of actions), and the **plan cost** is:

$$c(\pi) = \sum_{i=1}^{n} c(a_i)$$

A **cost-optimal solution** is a plan π^* that minimizes cost among all solutions:

$$c^* := c(\pi^*) = \min\{c(\pi') \mid \pi' \text{ is a solution for } \mathcal{P}\}$$

Numerical Example

In our 2-city world, if s_0 = truck-at-A and S_g = {truck-at-B}, the only plan is $\pi = \langle$drive-A-to-B$\rangle$ with cost $c(\pi) = 120$. This is also optimal: $c^* = 120$.
If the goal were S_g = {truck-at-A} (start and end at A), the only plan is $\pi = \langle$drive-A-to-B, drive-B-to-A$\rangle$ with $c(\pi) = 120 + 130 = 250$.

1.3 The Planning Graph

The planning domain Σ induces a **directed weighted graph** $\mathcal{G}_\Sigma = (\mathcal{V}, \mathcal{E}, w)$:

- $\mathcal{V}$: one vertex per state $s \in S$.
- $\mathcal{E}$: a directed edge $e = (s, \gamma(s, a))$ for every state s and applicable action a. Each edge represents one action application.
- $w(e) = c(a)$: the edge weight equals the action cost. For convenience, the paper writes $c(e)$ instead of $w(e)$.

Key Idea

Planning = graph search. A plan π corresponds to a path from s_0 to some goal state in $\mathcal{G}_\Sigma$. An optimal plan corresponds to a shortest (least-cost) path. This is why algorithms like A^* are directly applicable to planning.

1.4 A Quick Reminder: A^* Search

A^* is the standard algorithm for finding cost-optimal paths in weighted graphs. It maintains:

- $g(s)$: the cost of the best path found so far from s_0 to s.
- $h(s)$: a **heuristic** estimate of the remaining cost from s to a goal.
- $f(s) = g(s) + h(s)$: the estimated total cost of a path through s.

A^* always expands the node with the smallest $f(s)$ from its OPEN list. With a **consistent** (monotone) heuristic h, A^* is guaranteed to find a cost-optimal path.

Consistency Reminder

A heuristic h is **consistent** if for every edge $e = (s, s')$ with cost $c(e)$: $h(s) \leq c(e) + h(s')$. Consistency implies admissibility ($h(s) \leq h^*(s)$, the true cost-to-go). In the paper, heuristic consistency plays a key role in the theoretical guarantees of ACE.

2 The Problem: Multiple Action-Cost Estimates (P-MACE)

This section introduces the paper's central contribution: a new planning problem formulation where action costs are not known exactly but can be estimated with increasing precision at increasing computational expense.

2.1 Motivation: Why Action Costs Can Be Expensive to Compute

In standard (textbook) planning, action costs are given in the problem description and computing them is essentially free. But in many real-world applications, this assumption breaks down:

- **Robot motion planning**: Computing the true cost of a movement requires checking geometric and kinematic constraints, computing collision-free paths, and integrating over continuous trajectories. This can take significant CPU time.
- **Transportation logistics**: The travel time between two cities depends on traffic, weather, road conditions, speed limits, elevation, and curvature. A quick estimate uses fixed average speed, but an accurate estimate requires querying an external service (e.g., Google Maps), which takes 10+ ms per query.
- **External action models**: Some planners call external "black box" models to compute costs, where each call has non-trivial latency.

Key Idea

The paper's central insight: action-cost computation can be *spread across multiple estimators*, starting with a quick-but-rough estimate and progressively refining it only when needed. This avoids wasting computation on actions that the planner will never use.

2.2 Cost Estimator Functions

Definition 2.1 (Cost Estimators Function Θ)**.** The **cost estimators function** Θ for a planning domain Σ maps each edge $e \in \mathcal{E}$ (equivalently, each ground action in context) to a finite, non-empty, **ordered** set of estimators:

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

subject to: for all e, for all i,

$$\theta_e^i : \emptyset \to \mathbb{R}^+ \times \mathbb{R}^+$$

What this says in plain English: Every ground action (edge in the planning graph) comes equipped with a list of estimators, numbered $1, 2, \ldots, k(e)$. Each estimator θ_e^i, when called, returns two numbers: a lower bound and an upper bound on the true cost of edge e. The estimators are ordered by *increasing estimation time*: the first estimator is cheapest (and usually least accurate), and subsequent estimators are progressively more expensive (and more accurate).

Symbol-by-symbol breakdown:

- $e \in \mathcal{E}$ — a specific edge (ground action) in the planning graph.
- $k(e) \in \mathbb{N}$ — the number of estimators available for edge e. Different edges can have different numbers of estimators.
- θ_e^i — the i-th estimator for edge e. It is a function (or procedure) that can be called to obtain bounds on $c(e)$.
- $\mathbb{R}^+ \times \mathbb{R}^+$ — each estimator returns a pair of non-negative real numbers: $(c_{min}^i(e),\ c_{max}^i(e))$, the lower and upper bounds.
- The ordering $(\theta_e^1, \theta_e^2, \ldots)$ is by increasing estimation time: θ_e^1 is fastest, $\theta_e^{k(e)}$ is slowest.

Transportation Logistics Estimators

Consider the action "drive from Toronto to Ottawa." The true cost (travel time) is, say, 4.5 hours. Three estimators might be available:

i	Estimator θ_e^i	c_{min}^i	c_{max}^i	Time
1	Fixed distance / max speed	3.0 h	12.0 h	<1 ms
2	Average speed by road type	3.5 h	7.0 h	5 ms
3	Google Maps API query	4.0 h	5.0 h	50 ms

Estimator 1 is nearly free: it just divides the straight-line distance by the maximum speed limit (lower bound) and by a very slow speed (upper bound). The interval $[3.0, 12.0]$ is wide but cheap. Estimator 3 queries an external service and returns a tight interval $[4.0, 5.0]$, but it costs 50 ms — and in a plan with hundreds of actions, this adds up.

2.3 Assumptions About Estimators

The paper makes three assumptions about the estimators, which are crucial for the theoretical guarantees:

1. **Finite bounds.** Each estimator provides finite bounds on the true cost:

$$0 \leq c_{min}^i(e) \leq c(e) \leq c_{max}^i(e) < \infty$$

   The true cost $c(e)$ always lies within the interval $[c_{min}^i(e), c_{max}^i(e)]$.

2. **Positive costs.** Whenever $c(e) > 0$, the lower bound is also positive: $c_{min}^i(e) > 0$. Zero-cost actions are identified exactly: $c(e) = 0$ iff $c_{min}^i(e) = 0$.

3. **Monotone tightening.** Later (more expensive) estimators produce intervals that are *nested* within earlier (cheaper) ones:

$$[c_{min}^j(e),\ c_{max}^j(e)] \subseteq [c_{min}^i(e),\ c_{max}^i(e)], \quad \forall\, e,\ \forall\, i < j$$

Why Monotone Tightening Matters

Assumption 3 is what makes the "incremental refinement" idea work. Each call to a more expensive estimator can only narrow the interval — it never makes things worse. The lower bound can only go up (or stay), and the upper bound can only go down (or stay). This means we can stop refining whenever the current interval is "good enough" for our planning needs.

Numerical Example

Continuing the Toronto–Ottawa example, the nesting is:

$$[4.0, 5.0] \subseteq [3.5, 7.0] \subseteq [3.0, 12.0]$$

Each subsequent estimator narrows the interval. If the planner finds that the wide interval $[3.0, 12.0]$ is sufficient to decide that this action is or is not part of an optimal plan, it never needs to call the expensive Google Maps API.

2.4 The P-MACE Problem

Definition 2.2 (P-MACE Problem)**.** A **Planning with Multiple Action Cost Estimates (P-MACE)** problem is a tuple:

$$\mathcal{P} = (\Sigma, \Theta, s_0, S_g)$$

where Σ is a planning domain with cost function $c \in \Sigma$ *unknown*, Θ is a cost estimators function for Σ, $s_0 \in S$ is an initial state, and $S_g \subset S$ is a set of goal states.

What is new compared to classical planning?

In classical planning, the cost function c is known exactly. In P-MACE, c is *unknown* — the planner can only access it through the estimators in Θ. Each estimator call returns an interval bounding the true cost, and more expensive estimators give tighter intervals.

Definition 2.3 ($\mathcal{B}$-Optimal Solution)**.** A **$\mathcal{B}$-optimal solution** to a P-MACE problem $\mathcal{P}$ is a plan $\pi^{\mathcal{B}}$ such that:

$$c(\pi^{\mathcal{B}}) \leq c^* \times \mathcal{B}$$

where c^* is the (unknown) optimal cost and $\mathcal{B} \geq 1$.

What it says: A $\mathcal{B}$-optimal plan has cost at most $\mathcal{B}$ times the true optimal cost. When $\mathcal{B} = 1$, we require true optimality (but this may require evaluating all estimators). When $\mathcal{B} > 1$, we accept a plan that may be suboptimal but is guaranteed to be within a factor of $\mathcal{B}$ of optimal.

Numerical Example

Suppose the true optimal plan has cost $c^* = 100$ (minutes).

- A 1-optimal solution must have cost ≤ 100 (truly optimal).
- A 1.5-optimal solution must have cost ≤ 150 (at most 50% worse).
- A 2-optimal solution must have cost ≤ 200 (at most twice as bad).

Allowing $\mathcal{B} > 1$ gives the planner room to skip expensive estimations: if the planner can certify that a plan's cost is within a factor of $\mathcal{B}$ of optimal using only cheap estimators, it need not refine further.

The Cost c^* Is Unknown!

A subtle but critical point: c^* itself is unknown because the true costs are unknown. The planner cannot directly check whether $c(\pi) \leq c^* \times \mathcal{B}$. Instead, it must use the available estimates (intervals) to reason about whether the condition is met. This is what makes the problem challenging.

2.5 Theorem 1 (Generality): P-MACE Generalizes Classical Planning

Theorem 2.1 (Generality)**.** *P-MACE problems are a generalization of classical planning problems.*

What it says: Every classical planning problem can be expressed as a P-MACE problem. P-MACE is strictly more general.

Proof idea: Take any classical planning problem where all costs $c(e)$ are known. Set $k(e) = 1$ for every edge (one estimator each), and let that estimator return $c^1_{min}(e) = c^1_{max}(e) = c(e)$ (the exact cost, with zero uncertainty). The resulting P-MACE problem has no relaxation for plan optimality (which means that $\mathcal{B} = 1$), so finding a $\mathcal{B}$-optimal solution is equivalent to finding a truly optimal solution, exactly as in classical planning.

Key Idea

Classical planning is the special case of P-MACE where every action has exactly one estimator that returns the exact cost. P-MACE adds two generalizations: (1) costs can be uncertain (bounded by intervals), and (2) there can be multiple estimators per action with varying accuracy and computational cost.

3 Plan Cost Uncertainty Quantification

Before describing the algorithms, we need a way to measure how uncertain a plan's cost is. This section formalizes the key quantities: plan cost bounds and the cost uncertainty ratio.

3.1 Valid Estimator Subsets and Plan Cost Bounds

Let Θ_Σ be the set of *all* estimators for all edges in $\mathcal{G}_\Sigma$. A subset $\Phi \subseteq \Theta_\Sigma$ is called **valid** if it contains at least one estimator per edge.

Given a plan $\pi = \langle a_1, \ldots, a_n \rangle$ and a valid Φ, the **plan lower bound** and **plan upper bound** w.r.t. Φ are:

$$c^\Phi_{min}(\pi) := \sum_{i=1}^{n} c^i_{min,\Phi}, \quad c^\Phi_{max}(\pi) := \sum_{i=1}^{n} c^i_{max,\Phi} \tag{Eq. 1}$$

Symbol-by-symbol breakdown:

- $c^i_{min,\Phi}$ — the *tightest* (highest) lower bound available in Φ for the i-th action a_i of plan π.
- $c^i_{max,\Phi}$ — the *tightest* (lowest) upper bound available in Φ for a_i.
- The sums accumulate these bounds over all n actions in the plan.

What it says: The plan cost lower bound is the sum of the tightest lower bounds for each action. The plan cost upper bound is the sum of the tightest upper bounds. The true plan cost lies between these: $c^\Phi_{min}(\pi) \leq c(\pi) \leq c^\Phi_{max}(\pi)$.

Numerical Example

A 3-action plan with tightest available bounds:

Action	$c^i_{min,\Phi}$	$c(a_i)$ (true, unknown)	$c^i_{max,\Phi}$
a_1	10	12	15
a_2	20	25	40
a_3	5	5	5

Plan cost bounds: $c^\Phi_{min}(\pi) = 10 + 20 + 5 = 35$, $c^\Phi_{max}(\pi) = 15 + 40 + 5 = 60$.
True plan cost: $c(\pi) = 12 + 25 + 5 = 42 \in [35, 60]$.
Note that a_3 has been estimated exactly ($c_{min} = c_{max} = 5$), contributing zero uncertainty. The uncertainty comes from a_1 (interval width 5) and especially a_2 (interval width 20).

3.2 The Cost Uncertainty Ratio $\eta(\pi)$

$$\eta(\pi) := \frac{\sum_{i=1}^{n} c^i_{max,\Phi}}{\sum_{i=1}^{n} c^i_{min,\Phi}} = \frac{c^\Phi_{max}(\pi)}{c^\Phi_{min}(\pi)} \tag{Eq. 2}$$

(If $\sum c^i_{min} = 0$, then $\eta(\pi) = 1$ by convention, since this implies $c(\pi) = 0$ — no uncertainty.)

What it computes: The ratio of the plan's upper bound to its lower bound. This single number captures the overall uncertainty in the plan's cost.

Interpretation:

- $\eta(\pi) = 1$: the cost is known exactly (upper = lower bound). Zero uncertainty.
- $\eta(\pi) = 2$: the true cost could be anywhere from c_{min} to $2 \times c_{min}$. The plan cost is known only up to a factor of 2.
- $\eta(\pi) = 4$: very uncertain — the cost could be up to 4 times the lower bound.

Numerical Example

From the previous example:

$$\eta(\pi) = \frac{60}{35} \approx 1.71$$

The plan cost is known up to a factor of about 1.71.
If we call a more expensive estimator for a_2 and get tighter bounds $[22, 30]$:

$$\eta(\pi) = \frac{15 + 30 + 5}{10 + 22 + 5} = \frac{50}{37} \approx 1.35$$

The uncertainty ratio has decreased. If we further refine a_1 to $[11, 13]$:

$$\eta(\pi) = \frac{13 + 30 + 5}{11 + 22 + 5} = \frac{48}{38} \approx 1.26$$

Each refinement reduces $\eta(\pi)$, approaching 1 as all estimates converge to the true costs.

Why $\eta(\pi)$ Matters for $\mathcal{B}$-Optimality

The cost uncertainty ratio $\eta(\pi)$ is directly connected to the ability to certify $\mathcal{B}$-optimality. If $\eta(\pi) \leq \mathcal{B}$, and π is an optimal plan w.r.t. the lower bounds, then π is guaranteed to be $\mathcal{B}$-optimal (by Theorem 2, below). So the planner's goal is to find plans where η is small enough relative to the target $\mathcal{B}$.

3.3 The Central Inequality

A fundamental property connecting the cost bounds, the uncertainty ratio, and the true plan cost:

$$c^\Phi_{min}(\pi) \leq c(\pi) \leq c^\Phi_{max}(\pi) = c^\Phi_{min}(\pi) \times \eta(\pi) \qquad \text{(Eq. 3)}$$

What it says: The true plan cost is sandwiched between the lower and upper bounds, and the upper bound equals exactly $\eta(\pi)$ times the lower bound. This is immediate from the definition of η.

Why it matters: If π is an optimistic plan (its lower bound equals the smallest possible lower bound over all plans), then:

$$c(\pi) \leq c^\Phi_{min}(\pi) \times \eta(\pi) \leq c^* \times \eta(\pi)$$

So if $\eta(\pi) \leq \mathcal{B}$, then $c(\pi) \leq c^* \times \mathcal{B}$, meaning π is $\mathcal{B}$-optimal.

3.4 Optimal Optimistic and Pessimistic Plans

Two special plans are important in the analysis:

- An **optimal optimistic plan** π_{opt} minimizes c^{Φ}_{min}: it is the plan that looks cheapest when you use the most favorable (lowest) cost bounds. Formally, $c^{\Phi}_{min}(\pi_{opt}) = c^{*\Phi}_{min}$, the optimal plan cost lower bound.
- An **optimal pessimistic plan** π_{pes} minimizes c^{Φ}_{max}: it is the plan that looks cheapest even under the most unfavorable (highest) cost bounds. Formally, $c^{\Phi}_{max}(\pi_{pes}) = c^{*\Phi}_{max}$, the optimal plan cost upper bound.

Numerical Example

Suppose there are two candidate plans and the available bounds are:

Plan	c^{Φ}_{min}	c^{Φ}_{max}	η
π_1	30	50	1.67
π_2	35	42	1.20

π_1 is the optimal optimistic plan (lowest lower bound: 30). π_2 is the optimal pessimistic plan (lowest upper bound: 42). Neither dominates the other: π_1 might be cheaper (if its true cost is near 30) or more expensive (if near 50).

3.5 Theorem 2 (Bound): Cost of an Optimal Optimistic Plan

Theorem 3.1 (Bound)**.** *Given a P-MACE problem $\mathcal{P}$, an optimal optimistic plan π_{opt} w.r.t. any valid $\Phi \subseteq \Theta_{\Sigma}$ satisfies:*

$$c(\pi_{opt}) \leq c^* \times \eta(\pi_{opt}) \qquad \text{(Eq. 4)}$$

What it says in plain English: The true cost of the most optimistic plan is at most $\eta(\pi_{opt})$ times the true optimal cost. So if the uncertainty ratio is low (say, $\eta = 1.2$), the optimistic plan's cost is within 20% of optimal.

Proof sketch:

1. Since π_{opt} has the lowest lower bound among all plans: $\sum c^i_{min} \leq c^*$ (because the lower bound of any plan, including the truly optimal one, is at most its true cost, and π_{opt} minimizes these lower bounds). This gives us Inequality (5) in the paper:

$$\sum_{i=1}^{n} c^i_{min} \leq c^* \qquad (5)$$

2. The true cost of π_{opt} is bounded by its upper bound: $c(\pi_{opt}) \leq \sum c^i_{max}$.
3. From Eq. 3: $c(\pi_{opt}) \leq \sum c^i_{max} = \sum c^i_{min} \times \eta(\pi_{opt})$. More precisely, each $c^i_{max} \leq c^i_{min} \times \eta(\pi_{opt})$ does *not* hold action-by-action, but the sum inequality follows: $\sum c^i_{max} = c^{\Phi}_{min}(\pi_{opt}) \times \eta(\pi_{opt})$, so $c(\pi_{opt}) \leq c^* \times \eta(\pi_{opt})$.

Corollary 3.2 (from Theorem 2)**.** *An optimal optimistic plan π_{opt} w.r.t. any valid Φ with $\eta(\pi_{opt}) \leq \mathcal{B}$ is $\mathcal{B}$-optimal.*

What this means for algorithm design: To find a $\mathcal{B}$-optimal plan, it suffices to find an optimal optimistic plan with $\eta \leq \mathcal{B}$. The planner can achieve this by selectively refining estimates (calling more expensive estimators) until either η drops below $\mathcal{B}$, or it determines that this is not achievable.

Numerical Example

Recall the plan π_1 from before with $c^{\Phi}_{min} = 30$, $\eta = 1.67$. Suppose the true optimal cost is $c^* = 32$. Then:

$$c(\pi_1) \leq c^* \times \eta(\pi_1) = 32 \times 1.67 = 53.4$$

If our target is $\mathcal{B} = 2$, we need $\eta(\pi_1) \leq 2$. Since $1.67 \leq 2$, the plan is certified as 2-optimal. But for $\mathcal{B} = 1.5$, we need $\eta \leq 1.5$, and $1.67 > 1.5$, so we need more refinement.

4 ACE: A^* with Cost Estimation

This section presents the paper's main algorithm, ACE, which is a generalization of A^* that handles P-MACE problems. ACE selectively calls estimators during search to find $\mathcal{B}$-optimal plans while minimizing estimation cost.

4.1 High-Level Idea

ACE works like A^*, but with a crucial difference: instead of using known edge costs to compute $g(s)$, ACE uses cost *estimates* that it incrementally refines during search. The key decisions ACE makes are:

1. **When to refine:** ACE calls a more expensive estimator for an edge only when the current uncertainty is too high (exceeds the target $\mathcal{B}$).
2. **When to stop refining:** ACE stops calling estimators for an edge when either (a) η drops to $\mathcal{B}$ or below, (b) a cheaper path to the same node has been found, or (c) all estimators have been exhausted.
3. **What costs to use for search:** ACE uses accumulated lower bounds g_{min} and upper bounds g_{max} instead of a single g value. The heuristic h is computed using the loosest (most optimistic) lower bounds, preserving consistency.

ACE's Two Deviations from A^*

1. **Heuristic computation**: A^* uses exact edge costs for heuristic computation. ACE uses the *loosest* (lowest) lower bound for each edge. This ensures the heuristic remains consistent even when costs are uncertain.
2. **g-value computation**: Instead of a single $g(s)$, ACE maintains $g_{min}(s)$ and $g_{max}(s)$ — accumulated lower and upper cost bounds along the path to s. An estimation loop refines these bounds during search.

4.2 The ACE Algorithm (Algorithm 1)

Here is a detailed walkthrough of ACE, presented line by line.

Algorithm 1: ACE
Input: Problem $\mathcal{P} = (\Sigma, \Theta, s_0, S_g)$, target $\mathcal{B}$
Parameter: Heuristic h, procedure GetEstimator
Output: Plan π, bound η

```
1:  g_min(s_0) ← 0;  g_max(s_0) ← 0
2:  OPEN ← ∅;  CLOSED ← ∅
3:  Insert s_0 into OPEN with f(s_0) = h(s_0)
4:  while OPEN ≠ ∅ do
5:    n ← best node from OPEN
6:    if Goal(n) then
7:      return trace(n), g_max(n)/g_min(n)
8:    Insert n into CLOSED
9:    for each successor s of n do
10:     if s not in OPEN ∪ CLOSED then
11:       g_min(s) ← ∞
12:     η ← ∞;  g ← g_min(n)
13:     θ ← GetEstimator(e = (n, s))
14:     while η > B and g < g_min(s) and θ ≠ ∅ do
15:       c, c̄ ← apply(θ)
16:       g ← g_min(n) + c;  ḡ ← g_max(n) + c̄
17:       η ← ḡ/g
18:       θ ← GetEstimator(e)
19:     if g < g_min(s) then
20:       g_min(s) ← g;  g_max(s) ← ḡ
21:       if s in OPEN ∪ CLOSED then
22:         Remove s from OPEN and CLOSED
23:       Insert s into OPEN with f(s) = g_min(s) + h(s)
24: return ∅, ∞
```

Detailed line-by-line explanation:

Lines 1–3: Initialization. The start state s_0 has zero accumulated cost ($g_{min} = g_{max} = 0$ because no actions have been taken yet). It is inserted into the OPEN list with priority $f(s_0) = 0 + h(s_0) = h(s_0)$, just like A^*.

Lines 4–5: Main loop. Like A^*, ACE repeatedly selects the node n with the smallest f-value from OPEN.

Lines 6–7: Goal test. When a goal state is reached, ACE returns the plan (by tracing back the path) and the achieved uncertainty ratio $\eta = g_{max}(n)/g_{min}(n)$.

Line 8: CLOSED list. Node n is added to CLOSED after expansion, as in A^*.

Lines 9–10: Successor generation. For each successor s of n, if s has never been seen before, its g_{min} is initialized to ∞ (line 11).

Lines 12–13: Estimation setup. η is initialized to ∞ (worst case). The variable $\underline{g}$ tracks the accumulated lower-bound cost to s via n. The first available estimator θ for edge $e = (n, s)$ is fetched.

Lines 14–18: The estimation loop (the core novelty). This is where ACE differs fundamentally from A^*. The loop iterates over estimators for the edge (n, s), calling progressively more expensive estimators, until one of three stopping conditions is met:

1. $\eta \leq \mathcal{B}$: The cost uncertainty ratio for the path through n to s is within the target bound. No need to refine further.
2. $\underline{g} \geq g_{min}(s)$: A previously found path to s already has a lower (or equal) lower-bound cost. There is no point in refining the current path further — it is already dominated.
3. $\theta = \emptyset$: All estimators for edge e have been exhausted.

Inside the loop (lines 15–18):

- Line 15: Apply the current estimator, getting lower bound $\underline{c}$ and upper bound $\bar{c}$ for edge e.
- Line 16: Accumulate: $\underline{g} = g_{min}(n) + \underline{c}$ is the path lower bound, $\bar{g} = g_{max}(n) + \bar{c}$ is the path upper bound.
- Line 17: Compute the uncertainty ratio of the path to s via n.
- Line 18: Fetch the next (more expensive) estimator.

Lines 19–23: Updating g-values. If the path through n provides a better (lower) lower-bound cost to s than any previously found path, update the g-values and (re-)insert s into OPEN. If s was already in OPEN or CLOSED, it is removed first (allowing reopening, which is necessary because bounds may have changed).

Line 24: No solution. If OPEN empties without finding a goal, no plan exists.

ACE Uses Lower Bounds for f-values

The priority function $f(s) = g_{min}(s) + h(s)$ uses the *lower bound* $g_{min}(s)$, not the true cost (which is unknown) or the upper bound. This is optimistic: ACE explores nodes that *might* be on a cheap path, even if the true cost turns out to be higher. This is exactly what A^* does, but with estimated lower bounds instead of exact costs.

ACE Walkthrough on a Small Graph

Consider a graph with states $\{s_0, s_1, s_2, s_g\}$ and edges:

Edge	True cost	θ^1 bounds	θ^2 bounds	θ^3 bounds
(s_0, s_1)	5	$[2, 20]$	$[4, 10]$	$[5, 5]$
(s_0, s_2)	8	$[6, 12]$	$[7, 9]$	$[8, 8]$
(s_1, s_g)	3	$[1, 12]$	$[2, 6]$	$[3, 3]$
(s_2, s_g)	2	$[1, 8]$	$[2, 2]$	—

The optimal plan is $s_0 \to s_1 \to s_g$ with true cost $5 + 3 = 8$. Let $h = 0$ (trivial heuristic) and $\mathcal{B} = 2$.

Step 1: Expand s_0 (only node in OPEN).

Process successor s_1: Edge (s_0, s_1).

- Call θ^1: $\underline{c} = 2, \bar{c} = 20$. $\underline{g} = 0 + 2 = 2$, $\bar{g} = 0 + 20 = 20$, $\eta = 20/2 = 10 > \mathcal{B} = 2$.
- Call θ^2: $\underline{c} = 4, \bar{c} = 10$. $\underline{g} = 0 + 4 = 4$, $\bar{g} = 0 + 10 = 10$, $\eta = 10/4 = 2.5 > 2$.
- Call θ^3: $\underline{c} = 5, \bar{c} = 5$. $\underline{g} = 0 + 5 = 5$, $\bar{g} = 0 + 5 = 5$, $\eta = 5/5 = 1.0 \leq 2$. Stop.

Set $g_{min}(s_1) = 5$, $g_{max}(s_1) = 5$, $f(s_1) = 5$.

Process successor s_2: Edge (s_0, s_2).

- Call θ^1: $\underline{c} = 6, \bar{c} = 12$. $\underline{g} = 0 + 6 = 6$, $\bar{g} = 0 + 12 = 12$, $\eta = 12/6 = 2.0 \leq 2$. Stop.

Set $g_{min}(s_2) = 6$, $g_{max}(s_2) = 12$, $f(s_2) = 6$.

Step 2: OPEN $= \{s_1(f = 5), s_2(f = 6)\}$. Expand s_1.

Process successor s_g: Edge (s_1, s_g).

- Call θ^1: $\underline{c} = 1, \bar{c} = 12$. $\underline{g} = 5 + 1 = 6$, $\bar{g} = 5 + 12 = 17$, $\eta = 17/6 \approx 2.83 > 2$.

- Call θ^2: $\underline{c} = 2, \bar{c} = 6$. $\underline{g} = 5 + 2 = 7$, $\bar{g} = 5 + 6 = 11$, $\eta = 11/7 \approx 1.57 \leq 2$. Stop.

Set $g_{min}(s_g) = 7$, $g_{max}(s_g) = 11$, $f(s_g) = 7$.
Step 3: OPEN $= \{s_2(f = 6), s_g(f = 7)\}$. Expand s_2.
Process successor s_g: Edge (s_2, s_g).

- Call θ^1: $\underline{c} = 1, \bar{c} = 8$. $\underline{g} = 6 + 1 = 7$, $\bar{g} = 12 + 8 = 20$, $\eta = 20/7 \approx 2.86 > 2$.
- Check: $\underline{g} = 7 \geq g_{min}(s_g) = 7$. Stop (condition 2).

Path through s_2 does not improve $g_{min}(s_g)$, so s_g is not updated.
Step 4: OPEN $= \{s_g(f = 7)\}$. Expand s_g — it is a goal! Return plan $s_0 \to s_1 \to s_g$ with $\eta = g_{max}(s_g)/g_{min}(s_g) = 11/7 \approx 1.57$.
Note: ACE found the plan $s_0 \to s_1 \to s_g$ with $\eta = 1.57 \leq 2 = \mathcal{B}$, certifying $\mathcal{B}$-optimality. The true cost of this plan is $5 + 3 = 8$, which is indeed the true optimal cost. ACE correctly found the optimum.
Also note that ACE used only 7 estimator calls total (3 for edge (s_0, s_1), 1 for (s_0, s_2), 2 for (s_1, s_g), and 1 for (s_2, s_g)), instead of the maximum possible 11 (if it had called all estimators for all edges).

4.3 The GetEstimator Procedure

Every call to GetEstimator(e) returns the *next* unused estimator from $\Theta(e)$, or $\emptyset$ if all estimators have been used.

Important: ACE's theoretical guarantees assume nothing about the *order* in which estimators are selected by GetEstimator. However, for practical efficiency, estimators should be ordered by increasing runtime. Each call to GetEstimator returns the next estimator θ_e^i with the shortest expected runtime among the unused ones. This is the approach taken in the experiments.

Estimation-Indifferent Planning as a Special Case

If GetEstimator is modified to immediately call *all* estimators for every edge and return only the tightest bounds, the result is an **estimation-indifferent** planner — the naive baseline that never skips any estimation. This is equivalent to classical A^* with costs equal to the tightest available bounds. ACE's advantage is that it often needs far fewer estimator calls than this baseline.

5 Analysis of ACE: Theoretical Guarantees

This section explains the four main theoretical properties of ACE: completeness, soundness (Φ set optimality), $\mathcal{B}$-soundness, and special $\mathcal{B}$-optimality.

5.1 Theorem 3 (Completeness): ACE Always Terminates

Theorem 5.1 (Completeness)**.** *ACE always terminates and returns a plan when one exists.*

What it says: ACE will not loop forever, and if a plan exists from s_0 to any goal state, ACE will find one.

Why it is true (proof sketch):

1. A^* always terminates because it processes a finite number of nodes. ACE inherits this: the graph has finitely many states (nodes).
2. Each edge has finitely many estimators, so the estimation loop (lines 14–18) terminates for each edge.

3. Every new node s encountered gets $g_{min}(s) = \infty$ initially (line 11), which is then set to a finite value by the first estimator (line 16, since $\underline{c}$ is always finite by Assumption 1). So s will be inserted into OPEN (line 23), ensuring it can eventually be expanded.
4. The condition in line 19 ($\underline{g} < g_{min}(s)$) ensures that OPEN insertions only happen when a strictly better path is found, which can happen at most finitely many times per node.

5.2 Lemma 1 (Φ Set Optimality)

Before proving soundness, we need a supporting result about what plan ACE returns.

Lemma 5.2 (Φ Set Optimality)**.** *Provided with a consistent heuristic h, ACE necessarily returns an optimal optimistic plan w.r.t. some $\Phi \subseteq \Theta_\Sigma$, if a plan exists.*

What it says: The plan ACE returns minimizes c^{Φ}_{min} (the sum of lower bounds) among all plans, where Φ is the set of all estimators that were actually invoked during the search.

Why it matters: This connects ACE's output to Theorem 2. Since the returned plan is optimal w.r.t. the lower bounds in Φ, if its uncertainty ratio η is at most $\mathcal{B}$, then by Theorem 2 (Corollary 1) the plan is $\mathcal{B}$-optimal.

How it follows from A^*'s properties: The proof relies on three facts:

1. Heuristic consistency is preserved when using lower bounds instead of true costs (Lemma 2 in the appendix — see below).
2. The set Φ consists of exactly the estimators invoked during search.
3. A^*'s optimality proof applies: with a consistent heuristic, whenever ACE expands a node, the path to that node has the optimal lower-bound cost.

5.3 Lemma 2 (Consistency Preservation)

Lemma 5.3 (Consistency Preservation)**.** *Given a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and two edge cost functions $\alpha : \mathcal{E} \to [0, \infty)$ and $\beta : \mathcal{E} \to [0, \infty)$ satisfying $\alpha(e) \leq \beta(e)$ for every $e \in \mathcal{E}$, we obtain the weighted digraphs $\mathcal{G}_\alpha = (\mathcal{V}, \mathcal{E}, \alpha)$ and $\mathcal{G}_\beta = (\mathcal{V}, \mathcal{E}, \beta)$. If h is a consistent parameterized heuristic, then h_α is consistent w.r.t. $\mathcal{G}_\beta$.*

What it says in plain English: If a heuristic h is a consistent parameterized heuristic, and we compute its values using lower edge costs α, then the resulting heuristic h_α remains consistent when applied to a graph with higher edge costs $\beta \geq \alpha$.

Why this matters for ACE: ACE uses the *loosest* (lowest) lower bounds to compute h-values. These lower bounds play the role of α, while the true (unknown) edge costs (or any tighter lower bounds encountered later) play the role of β. By this lemma, the heuristic h_α computed using the lowest available bounds is consistent w.r.t. the graph with higher costs $\beta \geq \alpha$. This ensures all of A^*'s optimality guarantees carry through.

Intuition: The key insight is about how a *parameterized* heuristic works. The heuristic h_α is computed using edge costs α. Consistency requires $h_\alpha(s) \leq \beta(e) + h_\alpha(s')$ for each edge $e = (s, s')$. Since h_α was consistent w.r.t. $\mathcal{G}_\alpha$, we know $h_\alpha(s) \leq \alpha(e) + h_\alpha(s')$. Because $\alpha(e) \leq \beta(e)$, we get $h_\alpha(s) \leq \alpha(e) + h_\alpha(s') \leq \beta(e) + h_\alpha(s')$, so the consistency condition is satisfied for $\mathcal{G}_\beta$ as well.

The Direction Matters

The lemma states: if h is a consistent parameterized heuristic and $\alpha(e) \leq \beta(e)$ for all edges, then h_α is consistent w.r.t. $\mathcal{G}_\beta$. The point is that if the heuristic is *computed* using lower costs α (the lower bounds), then it is consistent w.r.t. any graph with costs $\beta \geq \alpha$ (including the true costs). This is the correct direction for ACE: the heuristic is computed

using low costs and applied to search in a graph with higher (true) costs.

5.4 Theorem 4 ($\mathcal{B}$-Soundness)

Theorem 5.4 ($\mathcal{B}$-Soundness)**.** *Provided with a consistent heuristic h, ACE is $\mathcal{B}$-sound.*

What "$\mathcal{B}$-sound" means: An algorithm is **$\mathcal{B}$-sound** if every time it reports a plan as $\mathcal{B}$-optimal, the plan actually is $\mathcal{B}$-optimal (i.e., its true cost is at most $\mathcal{B} \times c^*$).

How this is shown (proof sketch): By induction on the order of nodes entering OPEN, starting from s_0. The key invariant is that each node n in OPEN satisfies $\eta(n) := g_{max}(n)/g_{min}(n)$ (where $g_{min}(n) = 0$ implies $\eta(n) = 1$). When ACE terminates at a goal state s_g, it returns $\eta(s_g) = g_{max}(s_g)/g_{min}(s_g)$, which is the uncertainty ratio for the path. If $\eta(s_g) \leq \mathcal{B}$, then by Lemma 1 (the plan is optimally optimistic) and Theorem 2 (the cost bound), the plan is $\mathcal{B}$-optimal.

ACE's Honesty

ACE always reports the achieved η alongside the plan. If $\eta \leq \mathcal{B}$, the user has a guarantee. If $\eta > \mathcal{B}$, ACE honestly reports this — the plan may still be good but is not *certified* to be $\mathcal{B}$-optimal. The post-search procedure ESE (Section 6) can then try to improve η.

5.5 Theorem 5 (Special $\mathcal{B}$-Optimality)

Theorem 5.5 (Special $\mathcal{B}$-Optimality)**.** *Given a P-MACE problem $\mathcal{P}$ with sub-optimality target $\mathcal{B}$, if every edge $e \in \mathcal{E}$ satisfies*

$$c^i_{max}(e)/c^i_{min}(e) \leq \mathcal{B}$$

for some i (that can be different for each edge), or $c^i_{min}(e) = c(e) = 0$, and a consistent heuristic h is used, then ACE is $\mathcal{B}$-optimal.

What it says in plain English: If, for every edge, there exists at least one estimator whose uncertainty ratio (max/min) is at most $\mathcal{B}$, then ACE is guaranteed to find a $\mathcal{B}$-optimal plan.

Why the condition matters: The condition ensures that it is always *possible* to achieve $\eta(\pi) \leq \mathcal{B}$ for any plan π, because each individual edge can be estimated within a factor of $\mathcal{B}$. The plan's η is a "weighted average" of the individual edge ratios, so if every edge ratio is at most $\mathcal{B}$, the plan ratio is also at most $\mathcal{B}$.

Corollary 5.6. *If the cost of every edge can ultimately be estimated exactly (i.e., for every $e \in \mathcal{E}$, there exists i such that $c^i_{min}(e) = c^i_{max}(e) = c(e)$), then $c^i_{max}(e)/c^i_{min}(e) = 1 \leq \mathcal{B}$, so ACE is $\mathcal{B}$-optimal for any $\mathcal{B} \geq 1$.*

*In the special case of **classical planning** (where costs are known exactly and each edge has one estimator returning the exact cost), this reduces to: ACE is optimal, just like A*.*

Numerical Example

Consider three edges with the following best-achievable uncertainty ratios:

Edge	Best c^i_{min}	Best c^i_{max}	Best ratio
e_1	10	10	1.0
e_2	20	25	1.25
e_3	5	9	1.8

For $\mathcal{B} = 2$: all ratios are ≤ 2, so ACE is guaranteed $\mathcal{B}$-optimal.
For $\mathcal{B} = 1.5$: edges e_1 and e_2 have ratios ≤ 1.5, but e_3 has ratio $1.8 > 1.5$. Theorem 5 does *not* apply. ACE might still find a 1.5-optimal plan (if the optimal plan does not use e_3, or if the plan-level η is still ≤ 1.5 even with e_3), but there is no guarantee.

6 Post-Search Improvement: End of Search Estimations (ESE)

ACE may terminate with $\eta(\pi) > \mathcal{B}$. This happens when the estimation loop exhausts all estimators for some edges without achieving the target uncertainty. The **End of Search Estimations (ESE)** procedure is a post-search step that tries to improve the plan's η by applying additional estimators to the edges in the returned plan.

6.1 The ESE Procedure (Procedure 1)

Procedure 1: End of Search Estimations (ESE)
Input: ACE's inputs and variables before termination
Parameter: Procedure GetEstimator
Output: Bound η

for each edge e that corresponds to an action from π **do**
  $\theta \leftarrow$ GetEstimator(e)
  while $\eta > \mathcal{B}$ **and** $\theta \neq \emptyset$ **do**
    $\underline{c}, \bar{c} \leftarrow$ apply(θ)
    update $c_{max}(\pi)$ using $\bar{c}$
    $\eta \leftarrow c_{max}(\pi)/c_{min}(\pi)$
    $\theta \leftarrow$ GetEstimator(e)
  if $\eta \leq \mathcal{B}$ **then**
    return η
return η

What it does in plain English: ESE iterates over the edges in the plan π found by ACE. For each edge, it calls additional estimators (that ACE did not use) to try to tighten the upper bound c_{max}. After each refinement, it recomputes η. If η drops to $\mathcal{B}$ or below, ESE stops and reports success.

Key design choices:

- ESE only updates *upper bounds* (c_{max}), not lower bounds. This is because the plan π was chosen as an optimal optimistic plan w.r.t. the lower bounds. Changing the lower bounds could invalidate π's optimality status, meaning a different plan might now be better. By only tightening upper bounds, π remains the same plan, and its cost guarantee can only improve.
- ESE can work with *any* remaining estimators, including expensive ones that ACE avoided during search. Since ESE only runs once (on the found plan), the cost is bounded by the number of edges in π times the number of remaining estimators.

Numerical Example

Suppose ACE found plan π with 3 edges and the following bounds:

Edge	c_{min} (from ACE)	c_{max} (from ACE)	Remaining estimators
e_1	10	15	θ^2, θ^3
e_2	20	40	θ^2
e_3	5	5	none

$c_{min}(\pi) = 35$, $c_{max}(\pi) = 60$, $\eta = 60/35 \approx 1.71$. Target: $\mathcal{B} = 1.5$.
ESE Step 1: Process e_1. Call θ^2: returns $[11, 12]$. Update c_{max} for e_1: 12 (tighter than 15). $c_{max}(\pi) = 12 + 40 + 5 = 57$, $\eta = 57/35 \approx 1.63 > 1.5$. Still not good enough.
ESE Step 2: Still on e_1. Call θ^3: returns $[11.5, 11.5]$ (exact). $c_{max}(\pi) = 11.5 + 40 + 5 = 56.5$, $\eta = 56.5/35 \approx 1.61 > 1.5$.
ESE Step 3: Process e_2. Call θ^2: returns $[22, 30]$. $c_{max}(\pi) = 11.5 + 30 + 5 = 46.5$, $\eta = 46.5/35 \approx 1.33 \leq 1.5$. Success! Return $\eta = 1.33$.

Why ESE Is Effective

ESE has a high success rate because ACE typically gets η very close to $\mathcal{B}$ (within a few percent). ESE then only needs a few additional estimator calls to push η below $\mathcal{B}$. In the paper's experiments, the mean normalized relative change in η due to ESE is only -5.72%, confirming that ACE leaves very little work for ESE to do.

7 Remark 1: Estimation-Indifferent Planning

An **estimation-indifferent** planner is the naive baseline: it calls *all* estimators for every edge and uses only the tightest bounds. This is equivalent to modifying Algorithm 1 to remove the condition $\eta > \mathcal{B}$ from line 14, so the estimation loop always runs to completion.

The estimation-indifferent approach is:

- **Complete and optimal**: It finds the best plan w.r.t. the tightest available bounds.
- **Expensive**: It calls every estimator for every edge, regardless of whether the information is needed. In transportation logistics, this means querying Google Maps for *every* possible route segment, even ones the planner will never use.

ACE's key advantage is that it typically uses far fewer expensive estimations. In the paper's experiments (Figure 1), ACE uses only 46–62% of the expensive estimations compared to the estimation-indifferent baseline, with no loss in solution quality.

8 Experimental Setup and Results

This section summarizes the experimental evaluation. The purpose is not to reproduce all results but to help you understand the experimental design and interpret the findings.

8.1 Benchmark Setup

The experiments use **PlanDEM** (Planning with Dynamically Estimated Action Models), a planner built by extending Fast Downward (a well-known planning system) to support cost estimators.

Benchmark problems: 20 domains from IPC (International Planning Competition) benchmarks, with 2 problems per domain, and various configurations of estimator uncertainty. For each problem:

- Each edge e with known PDDL cost $c_{PDDL}(e)$ is given 3 synthetic estimators:

$$\theta_e^1 : c_{min}^1 = 1 \times c_{PDDL}(e),\ c_{max}^1 = 4 \times c_{PDDL}(e) \quad \text{(ratio 4)}$$
$$\theta_e^2 : c_{min}^2 = 2 \times c_{PDDL}(e),\ c_{max}^2 = 4 \times c_{PDDL}(e) \quad \text{(ratio 2)}$$
$$\theta_e^3 : c_{min}^3 = 2 \times c_{PDDL}(e),\ c_{max}^3 = 2 \times c_{PDDL}(e) \quad \text{(exact, ratio 1)}$$

  The true cost is $c(e) = 2 \times c_{PDDL}(e)$.

- A parameter $p_1 \in \{0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ controls what fraction of edges require estimation (vs. having known exact costs).
- $\mathcal{B}$ ranges from 1 to 4 in steps of 0.25.

8.2 Key Findings

Finding 1: ACE uses significantly fewer expensive estimations. Even in the worst case ($p_1 = 1$, all actions estimated, $\mathcal{B} = 1$), ACE uses only 62% of the expensive estimators compared to the baseline. For $p_1 = 0.1$, this drops to 46%. The savings come from two sources:

- The $\eta > \mathcal{B}$ condition (line 14): when the path uncertainty is already within $\mathcal{B}$, no further estimation is needed.
- The $\underline{g} < g_{min}(s)$ condition (line 14): when a cheaper path to s already exists, the current path need not be refined.

Finding 2: Relaxing $\mathcal{B}$ yields rapid savings. Even slightly relaxing $\mathcal{B}$ from 1 (exact optimality) to 1.25 or 1.5 produces large reductions in the number of expensive estimations. The curves in Figure 1 drop steeply as $\mathcal{B}$ increases from 1.

Finding 3: Run-time savings can be dramatic. Figure 2 shows that when estimator calls are expensive (e.g., 0.1 seconds per call), the total planning time can be 10–100× lower with ACE compared to estimation-indifferent planning. When estimators are cheap ($< 10^{-4}$ seconds), the savings are negligible (the estimation time is dominated by the search time itself).

Finding 4: Success rate is 100% when $\mathcal{B}$ is achievable. Since each edge has an estimator with ratio 1 (exact cost), Theorem 5 guarantees $\mathcal{B}$-optimality for any $\mathcal{B} \geq 1$. The experiments confirm this: ACE always finds a $\mathcal{B}$-optimal plan.

8.3 Ontario Transportation Logistics Experiment

The paper includes a real-world experiment with transportation data from the Ontario province in Canada:

- 9 cities, 30 road segments, 2 trucks, 6 packages.
- Distances from Google Maps (100m resolution).
- Cost function: $c(a) = c_1 \times d + c_2 \times t$, where d is distance (km), t is time (minutes), $c_1 = 0.56$ USD/km, $c_2 = 0.5$ USD/minute.
- Two estimators for drive actions:
  1. **Cheap**: Assumes constant speed (20 km/h for bad traffic, 100 km/h for good traffic). Quick to compute.
  2. **Expensive**: Uses pessimistic and optimistic travel times from Google Maps for a specific day and time. Requires online query.

Results (Table 2): ACE saved 60–100% of costly estimator calls compared to estimation-indifferent planning. The best achievable η ranged from 8.5% to 53.8% (i.e., η between 1.085 and 1.538),

demonstrating that cost uncertainty is often unavoidable in practice. Assuming realistic query times (1–10 ms), ACE reduced planning time from 18 hours to 7 hours in the worst case.

9 Connecting the Pieces: How ACE Solves P-MACE

Here is the complete logical flow:

1. **Problem**: We have a P-MACE problem where action costs are unknown but can be estimated with increasing accuracy at increasing expense.
2. **Goal**: Find a plan whose cost is within a factor of $\mathcal{B}$ of optimal, while minimizing the computational cost of estimations.
3. **Key theoretical result** (Theorem 2 + Corollary 1): If we can find an optimal optimistic plan π_{opt} w.r.t. some set of estimators Φ, and if $\eta(\pi_{opt}) \leq \mathcal{B}$, then π_{opt} is $\mathcal{B}$-optimal.
4. **Algorithm** (ACE): A generalization of A^* that incrementally calls estimators during search, using the target $\mathcal{B}$ to decide when to stop refining. ACE returns an optimal optimistic plan w.r.t. the estimators it used (Lemma 1), together with the achieved η.
5. **Post-processing** (ESE): If ACE's $\eta > \mathcal{B}$, ESE applies additional estimators to the found plan's edges, tightening upper bounds until $\eta \leq \mathcal{B}$ or all estimators are exhausted.
6. **Guarantees**: ACE is complete (Theorem 3), $\mathcal{B}$-sound (Theorem 4), and $\mathcal{B}$-optimal when per-edge uncertainty can be bounded by $\mathcal{B}$ (Theorem 5).

The Fundamental Trade-off

P-MACE makes explicit a trade-off that is implicit in many real-world planning applications: **estimation accuracy vs. estimation cost**. Classical planning assumes this trade-off does not exist (costs are free and exact). P-MACE and ACE provide a principled framework for navigating this trade-off, with formal guarantees on solution quality.

10 Summary of Notation

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

Symbol	Type / Dimensions	Meaning
$\Sigma = (S, A, \gamma, c)$	4-tuple	Planning domain
S	Finite set	Set of states
A	Finite set	Set of ground actions
$\gamma : S \times A \to S$	Partial function	State-transition function
$c : S \times A \to \mathbb{R}^+$	Function	Action cost function (unknown in P-MACE)
$\mathcal{P} = (\Sigma, \Theta, s_0, S_g)$	Tuple	P-MACE problem
s_0	State $\in S$	Initial state
$S_g \subset S$	Set of states	Goal states
$\mathcal{G}_\Sigma = (\mathcal{V}, \mathcal{E}, w)$	Weighted digraph	Planning graph induced by Σ
$e = (s, s') \in \mathcal{E}$	Directed edge	An action application (edge in $\mathcal{G}_\Sigma$)
$c(e)$	Scalar ≥ 0	True (unknown) cost of edge e
$\pi = \langle a_1, \ldots, a_n \rangle$	Sequence	A plan (sequence of actions)
$c(\pi)$	Scalar ≥ 0	True cost of plan π
c^*	Scalar ≥ 0	Optimal plan cost
$\lvert\pi\rvert$	Integer	Plan length (number of actions)
Θ	Function	Cost estimators function
$\Theta(e) = (\theta_e^1, \ldots, \theta_e^{k(e)})$	Ordered set	Estimators for edge e
$k(e)$	Integer ≥ 1	Number of estimators for edge e
θ_e^i	Estimator (function)	The i-th estimator for edge e
$c_{min}^i(e)$	Scalar ≥ 0	Lower bound from i-th estimator
$c_{max}^i(e)$	Scalar ≥ 0	Upper bound from i-th estimator
Θ_Σ	Set	All estimators for all edges
$\Phi \subseteq \Theta_\Sigma$	Subset	A valid set of estimators
$c_{min}^\Phi(\pi)$	Scalar ≥ 0	Plan lower bound w.r.t. Φ
$c_{max}^\Phi(\pi)$	Scalar ≥ 0	Plan upper bound w.r.t. Φ
$c_{min}^{*\Phi}$	Scalar ≥ 0	Optimal plan lower bound
$c_{max}^{*\Phi}$	Scalar ≥ 0	Optimal plan upper bound
$\eta(\pi)$	Scalar ≥ 1	Cost uncertainty ratio of plan π
$\mathcal{B}$	Scalar ≥ 1	Target sub-optimality bound
$\pi^\mathcal{B}$	Plan	A $\mathcal{B}$-optimal solution
$g_{min}(s)$	Scalar ≥ 0	Accumulated cost lower bound to s (in ACE)
$g_{max}(s)$	Scalar ≥ 0	Accumulated cost upper bound to s (in ACE)
$f(s) = g_{min}(s) + h(s)$	Scalar ≥ 0	Priority value in ACE's OPEN list
$h(s)$	Scalar ≥ 0	Heuristic estimate of cost-to-go from s
$\underline{c}, \bar{c}$	Scalars ≥ 0	Lower/upper bound from one estimator call
$\underline{g}, \bar{g}$	Scalars ≥ 0	Accumulated bounds via current path
p_1	Scalar $\in [0, 1]$	Probability that an edge requires estimation
p_2, p_3	Scalars $\in [0, 1]$	Probabilities of 2nd/3rd estimator existence
τ	Scalar ≥ 0	Per-estimation run-time (in experiments)

11 Summary: Key Takeaways

1. **P-MACE is a natural generalization of classical planning.** It captures the common real-world situation where computing action costs is itself expensive and can be done at multiple levels of accuracy. Classical planning is the special case where costs are free and exact.

2. **The cost uncertainty ratio $\eta(\pi)$ is the key quantity.** It measures how much the plan's true cost could vary given the current estimates. Theorem 2 connects η directly to $\mathcal{B}$-optimality: $\eta \leq \mathcal{B}$ implies the plan is $\mathcal{B}$-optimal.

3. **ACE generalizes A^* to handle cost uncertainty.** It differs from A^* in two ways: (a) it uses lower bounds for heuristic computation (preserving consistency via Lemma 2), and (b) it incrementally refines cost estimates during search, stopping when the uncertainty is within the target $\mathcal{B}$ or when further refinement cannot help.

4. **ACE is complete, $\mathcal{B}$-sound, and $\mathcal{B}$-optimal under certain conditions.** It always terminates and finds a plan if one exists. The plans it reports as $\mathcal{B}$-optimal truly are $\mathcal{B}$-optimal. When every edge has an estimator with ratio $\leq \mathcal{B}$, it is guaranteed to find a $\mathcal{B}$-optimal plan.

5. **ESE provides a post-search safety net.** When ACE fails to achieve $\eta \leq \mathcal{B}$ during search, ESE can often close the gap with a few additional estimator calls on the found plan's edges.

6. **Experimental results are strong.** ACE saves 38–54% of expensive estimations compared to the naive baseline, with no loss in solution quality. On real-world transportation data, this translates to hours of saved computation time.