Graph Search and Automated Planning with Multiple Cost Estimators

Eyal Weiss

Department of Computer Science
Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University

Ramat-Gan, Israel October 2025

This work was carried out under the supervision of Prof. Gal A. Kaminka, Department of Computer Science, Bar-Ilan University.

To my beloved wife Yael for her love and support; to my two sons – Noam and Erez – who were both born during my PhD studies and have made this period even more special; to my parents who have raised me to be the man I am today; and to all my beloved family.

Last but not least, to my supervisor Gal for his endless patience, guidance, and clarity that have helped me overcome every challenge.

Acknowledgements

Practicing science has been a passion of mine and a source of happiness and comfort. I am grateful for everything in my life that has led me to pursue this path. This includes wonderful people who have supported me, offered guidance, and encouraged me to explore my curiosity. I cannot name them all, but I do wish to mention the most prominent people of whom I will forever be in debt.

My wife and partner for life, Yael, for her love, and for making me the man I am today. My kids, Noam and Erez, for the endless joy of being their father and for wanting to be a better person for them. My dog, Berta, for teaching me many lessons about what is really meaningful in life.

My parents, Uri and Dalia, who from a very young age gave me the confidence that I can achieve whatever I desire. My sister Michal for her love and encouragement. My brother Amir for providing an example of excellence and high moral at every step of this life. My grandparents from my father side, Menachem and Hanna, and from my mother side, Oded and Bracha, for their love, courage, and for providing an example of strong spirit.

My PhD supervisor, Gal, who has dedicated so much of his energy, time and skill to help me along every step of the way. I learned so much, and I still have a lot to learn from him. My MSc superivor, Michael, who helped ignite the flame of my love to research. My colleagues, Alexander Shleyfman and Ariel Felner, for providing crucial guidance where I needed it. My fellow students in the MAVERICK research group, for the fruitful conversations about life and

science. I also express my gratitude to the members of the ICAPS and SoCS communities for many lessons, insightful talks, and inspiration. Lastly, I thank the staff of Bar-Ilan University and especially the Computer Science Department for giving me a warm academic home.

I would like to conclude by thanking those who provided me with financial support: The Adams Fellowship Program of the Israel Academy of Sciences and Humanities, and by Bar-Ilan University's President Scholarship. My research was also partially funded by ISF Grant #2306/18 and BSF-NSF grant 2017764.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations and Acronyms

AI	Artificial Intelligence
PDDL	Planning Domain Definition Language
TAMP	Task and Motion Planning
IPC	International Planning Competition
EWDG	Estimated Weighted Digraph
EC	Energy Consumption
GDS^3P	Goal-Directed Single-Source Shortest Path
SLB	Shortest path with the Tightest Lower Bound
SUB	Shortest path with the Tightest Upper Bound
TASP	Tightest Admissible Shortest Path
UCS	Uniform Cost Search
w.r.t.	With respect to
w.l.o.g.	Without loss of generality
iff	If and only if

Abstract

The fields of graph search and automated planning have long relied on the simplifying assumption that the cost of applying an action or traversing an edge is known precisely and can be obtained at negligible computational expense. While this abstraction has enabled decades of theoretical progress and practical applications, it is increasingly mismatched with real-world problems, where costs are uncertain, expensive to compute, or available only through learned or external models. This dissertation addresses this fundamental gap by introducing new theoretical frameworks, problem formulations, and algorithms for planning and shortest-path search with multiple action- or edge-cost estimates.

The dissertation begins by establishing a formal background for reasoning about multiple estimators in graph search. It introduces **estimated weighted digraphs (EWDGs)** in which each edge is associated with a finite sequence of estimators that provide progressively tighter lower and upper bounds at increasing computational cost. This generalization allows classical search and planning problems to be reformulated in a way that explicitly accounts for estimation effort.

Building on this foundation, the dissertation develops formulations and methods for solving two central problems: finding paths that achieve the tightest possible lower bound on the optimal cost, and identifying paths that offer the strongest admissibility guarantees under estimator uncertainty. Together, these problems generalize the classical notion of shortest paths to settings where costs are only partially known and must be refined through computation. The work establishes the theoretical connections between these formulations, provides strategies for solving them optimally, and demonstrates empirically that rea-

soning explicitly about estimation effort can greatly reduce computational overhead while maintaining rigorous guarantees on solution quality.

The dissertation next broadens the scope from graph search to automated planning. It begins by introducing the paradigm of online modeling for offline planning, which challenges the traditional view of planning over a fixed, fully specified model. Instead, planning is treated as a process that dynamically invokes cost estimators as needed, thereby reasoning explicitly about the trade-off between computational effort and estimation accuracy.

Finally, the framework is instantiated in practical automated planning systems, where concrete planning algorithms that leverage multiple cost estimators for actions are suggested and implemented in a state-of-the-art planner. These algorithms can efficiently identify plans that meet specified suboptimality guarantees while attempting to minimize reliance on costly estimations. Through both synthetic benchmarks and a real-world transportation logistics problem, it is demonstrated that substantial savings in estimation effort and overall planning time can be achieved without sacrificing solution quality. This work provided the first end-to-end demonstration of planning systems that explicitly reason about cost-estimation time as part of the planning process.

Taken together, the contributions of this dissertation advance a unified theoretical and algorithmic foundation for shortest-path search and planning in settings where costs are uncertain, expensive, or incrementally estimable. By extending classical formulations to account for multiple estimators and their trade-offs, the work bridges the gap between elegant models of search and the realities of practical decision-making systems. Beyond their theoretical significance, these results are directly applicable to domains such as robotics, logistics, and autonomous systems, where estimation cost and accuracy are inseparable from the planning process.

In broad terms, this research opens the door to search and planning frameworks that are more expressive, resource-aware, and robust to uncertainty. By integrating cost estimation into the very definition of the planning problem, the dissertation redefines how optimality, admissibility, and efficiency are understood in modern AI search.

Chapter 1

Introduction

"Not everything that can be counted counts, and not everything that counts can be counted."

Albert Einstein

The shortest-path problem and its extension to planning are cornerstones of artificial intelligence. For decades, research has advanced under the simplifying assumption that action or edge costs are both known in advance and available at negligible computational expense. This assumption has enabled elegant algorithms such as Dijkstra's algorithm, A*, and their numerous descendants to power a wide range of applications in robotics, planning, scheduling, and optimization. However, the growing complexity of real-world systems increasingly renders this abstraction inadequate. In many modern domains, costs are uncertain, must be obtained through expensive computation or external queries, or are only available in approximate form through learned or data-driven models. This growing disconnect between the classical assumption and the realities of practical applications motivates a fundamental rethinking of search and planning. This is what is presented in this dissertation.

Section 1.1 begins by further motivating the need for methods that address the gap outlined above. Section 1.2 reviews the methods developed in this thesis. Finally, Section 1.3 lists publications resulting from this dissertation.

1.1 Motivation

Consider motion planning in robotics. To decide whether a robot arm can move along a certain trajectory, the planner must evaluate geometric feasibility, collision probabilities, and possibly dynamic constraints. Each of these evaluations may involve invoking costly simulators or physics-based models. In mobile robotics, travel time predictions may depend on terrain models, weather forecasts, or crowd density—all of which require external queries and substantial computation. The notion that such edge costs can be instantly retrieved as fixed scalars is no longer sustainable. Instead, it is common to rely on hierarchies of estimates: quick but coarse heuristics that provide loose bounds, refined estimations from more detailed models, and eventually highly accurate but time-consuming simulations. Planning under these conditions requires algorithms that not only search over paths or plans but also reason about which estimations to perform, when to perform them, and how to balance accuracy against computational effort.

A similar challenge arises in transportation and logistics. Routing a fleet of trucks, drones, or delivery robots requires estimating travel times and costs under uncertain conditions. A quick database lookup might provide distance-based estimates, while online services such as Google Maps can incorporate real-time traffic, and dedicated simulations can incorporate vehicle-specific constraints such as load or fuel consumption. Each estimator provides different levels of accuracy and requires different amounts of computational or monetary resources. Planners that ignore these costs may waste hours invoking expensive estimations unnecessarily, while planners that rely exclusively on coarse estimates risk producing poor solutions. A framework that systematically integrates multiple estimators can achieve both efficiency and reliability, yielding practical benefits in large-scale logistics operations.

The issue extends beyond robotics and logistics. In many AI systems, estimations of cost, risk, or reward are increasingly produced by machine learning models. These models vary in their computational expense and predictive accuracy. For example, in autonomous driving, a fast lightweight neural network

might give a rough estimate of collision risk, while a more accurate but slower model may refine this prediction. In natural language processing, one model may provide a cheap approximation of token likelihoods, while another, larger model offers better estimates at the cost of slower inference. As AI systems grow more complex, planners must coordinate between multiple predictive models, selectively invoking them to make the best use of computational resources. A generalized framework for reasoning about multiple estimators can provide the necessary theoretical foundation for these decisions.

Traditional search and planning methods are ill-suited to this reality because they treat estimator costs as external to the problem definition. In classical formulations, search runtime is decomposed into node expansions and queue operations, while edge costs are assumed to be readily available. In practice, the time to obtain a cost often dominates runtime. For instance, costs that are derived from online queries, which are increasingly more common, can be orders of magnitude slower than standard search operations. Ignoring this discrepancy leads to algorithms that spend vast amounts of time on unnecessary estimator calls, undermining their applicability to real-world systems where responsiveness is crucial.

Moreover, the presence of multiple estimators introduces a fundamental new trade-off: investing more computation can yield tighter bounds, but these may or may not change the ultimate decision. For example, if a rough estimate already shows that one path is inferior to another, there is little value in refining it further. Conversely, if two paths appear nearly equal under loose estimates, investing in tighter estimations may be critical for making the right choice. Designing algorithms that exploit this trade-off requires rethinking basic operations of search—such as node expansion, path comparison, and heuristic evaluation—so that they account for both estimation and search effort.

Another motivation stems from the need for robust guarantees in uncertain settings. In real-world decision-making, it is not enough to produce a plausible plan; one must often certify its quality relative to the unknown true optimum. This has long been formalized through notions such as admissibility and bounded suboptimality. However, when true costs are unknown and only

estimators are available, these guarantees must be redefined. The concept of admissibility must be extended to operate on bounds, not exact values, and algorithms must be designed to produce solutions with the tightest possible guarantees given the available estimators. This interplay between theory and practice is essential for deploying AI planning systems in safety-critical domains such as autonomous vehicles, robotics, and large-scale infrastructure.

The motivation for this line of research is therefore twofold. On the one hand, it addresses a pressing gap between classical planning theory and the needs of modern applications, where cost computations are neither free nor exact. On the other hand, it opens up a rich theoretical landscape of new problem formulations, algorithms, and guarantees. By treating estimation as a resource to be managed rather than ignored, we can design planning systems that are not only more efficient but also more aligned with the realities of the environments in which they operate.

Finally, this research contributes to a broader vision of AI systems that reason not only about external actions but also about their own computational processes. In doing so, it connects to emerging themes across AI and robotics, such as metareasoning, resource-bounded computation, and learning under uncertainty. Planning with multiple estimators can be viewed as a microcosm of these broader challenges: it requires balancing speed and accuracy, integrating heterogeneous sources of information, and making principled trade-offs under uncertainty. Developing the theoretical and algorithmic tools to meet these challenges is both scientifically compelling and practically necessary, forming the central motivation for the work presented in this dissertation.

1.2 Thesis Overview

The motivations outlined above highlight a growing gap between classical formulations of search and planning and the realities of modern applications, where action and edge costs are uncertain and often expensive to compute. This dissertation addresses that gap by reformulating fundamental search and planning problems to explicitly account for multiple cost estimators. Across

the chapters, the work introduces new problem definitions, adapts and extends well-known search techniques, and demonstrates both theoretical guarantees and practical benefits. Together, these contributions establish a unified framework for search and planning under estimation uncertainty.

The thesis is structurally composed of two parts: Part I addresses graph search and Part II addresses automated planning. We next briefly describe each chapter.

1.2.1 Part I

Graph Search with Multiple Cost Estimators: Background and Mathematical Setting

Chapter 2 lays the formal foundation for the dissertation's graph-search view of estimation. It introduces **estimated weighted digraphs (EWDGs)** with a cost-estimator function Θ that maps each edge to an ordered sequence of estimators whose bounds tighten as runtime increases, and formalizes shortest path with estimated weights (including the goal-directed setup). It then defines the canonical problems addressed throughout the thesis—**SLB** (tightest lower bound), **SUB** (tightest upper bound), and **TASP** (tightest admissible shortest path)—and explains their relationships and roles, including the reduction from TASP to SLB and SUB. The chapter closes by showing that **heuristics computed from lower-bound edge estimates preserve consistency**, establishing a safe baseline for informed search over EWDGs.

Optimally Solving Shortest Path Tightest Lower-Bound

Chapter 3 develops optimal algorithms for SLB, whose objective is to find a path achieving the tightest lower bound on the unknown optimal cost. It adapts classical best-first ideas to the multi-estimator regime via BEAUTY, which dynamically invokes estimators during search to keep expensive calls minimal, and A-BEAUTY, which iterates BEAUTY to focus later passes using bounds discovered earlier; both are proved correct and complete. The empirical study contrasts these methods with estimation-indifferent baselines (e.g., EI-UCS) and analyzes

variants (e.g., BEAUTY-PS, different estimator accuracies), demonstrating substantial reductions in costly estimator calls while achieving optimal SLB solutions.

Optimally Solving Tightest Admissible Shortest Path

In Chapter 4 the thesis turns to TASP, which seeks the solution with the tightest admissibility factor. The chapter introduces BEAST to compute SUB efficiently and then BEAUTY&BEAST, which exploits coupling between SLB and SUB to solve TASP more effectively than treating them independently. Experiments show sizable savings in expensive estimator usage (with high variability across domains) and quantify how SLB-derived information improves BEAST in pruning and overall efficiency when combined in BEAUTY&BEAST.

1.2.2 Part II

Online Modeling for Offline Planning

Chapter 5 motivates online modeling for offline planning in a broad perspective: instead of committing to a single, fully precomputed action model, the planner invokes model/parameter estimators on demand during planning, potentially at multiple fidelity levels. It critiques the traditional "offline modeling" pipeline (early commitment to modeling choices and computation; difficulty with learned, uncertain, or non-declarative models) and proposes a planning problem definition that explicitly includes action-model estimators with controllable accuracy/time trade-offs. The chapter also contrasts the projected cost of full offline modeling with the dynamic modeling time incurred only for actions actually explored, making the computational trade-off explicit.

Automated Planning with Multiple Cost Estimators

Building on the previous chapters, Chapter 6 integrates the framework into automated planning. It shows how to compute admissible, consistent heuristics cheaply from the loosest lower-bound estimates, and then sketches selective strategies that incorporate tighter estimates for heavily reused edges and couple

g- and *h*-value estimation when beneficial. The chapter presents planner-level algorithms (notably ACE) and their embedding in a practical system (PlanDEM), with results on IPC benchmarks and a logistics case study demonstrating large reductions in estimator calls while preserving guarantees and plan quality.

1.2.3 Discussion and Future Work

Chapter 7 reflects on the broader implications of the work, discusses limitations, and outlines promising directions for future research. Key points include the reliance on monotone estimator bounds, the trade-offs between search effort and estimator effort, and the need for richer estimator models that capture stochasticity and learning-based uncertainty. The chapter highlights practical considerations such as caching, distributed computation, and integration into robotics and logistics systems. More broadly, it emphasizes that estimation should be treated as a first-class element of the planning problem, and sketches avenues for extending the framework to hierarchical settings, multi-agent systems, and other complex domains. This closing chapter situates the dissertation within the larger trajectory of AI research, underscoring its contribution to more resource-aware, uncertainty-conscious decision-making systems.

1.3 Publications Resulting from this Research

Described here is the list of publications resulting from the research conducted as part of my doctoral thesis, summarized in this dissertation.

1. Eyal Weiss, Gal A. Kaminka. "Planning with Dynamically Estimated Action Costs" In *Workshop on Reliable Data-Driven Planning and Scheduling*. 2022.
2. Eyal Weiss, Gal A. Kaminka. "Position Paper: Online Modeling for Offline Planning" In *Workshop on Reliable Data-Driven Planning and Scheduling*. 2022.
3. Eyal Weiss, Gal A. Kaminka. "Planning with Multiple Action-Cost Estimates" In *Proceedings of the International Conference on Automated Planning*

and Scheduling: ICAPS 2023, pp. 427-437. 2023.

4. Eyal Weiss, Ariel Felner, Gal A. Kaminka. "A Generalization of the Shortest Path Problem to Graphs with Multiple Edge-Cost Estimates" In *Proceedings of the European Conference on Artificial Intelligence: ECAI 2023*, pp. 2607-2614. 2023.
5. Eyal Weiss, Ariel Felner, Gal A. Kaminka, "Tightest Admissible Shortest Path" In *Proceedings of the International Conference on Automated Planning and Scheduling: ICAPS 2024*, pp. 643-652, 2024.

Publications that are *not* part of the thesis but are based on research conducted during my doctoral studies:

- Daniel Gnad, Lee-or Alon, Eyal Weiss, Alexander Shleyfman, "PDBs Go Numeric: Pattern-Database Heuristics for Simple Numeric Planning" In *Proceedings of the AAAI Conference on Artificial Intelligence: AAAI 2025*, pp. 26507-26515, 2025.

Part I

Graph Search with Multiple Cost Estimators

Chapter 2

Graph Search with Multiple Cost Estimators: Background and Mathematical Setting

"There's no sense in being precise when you don't even know what you're talking about."

John von Neumann

2.1 Introduction

Finding the shortest path in a directed, weighted graph is fundamental to artificial intelligence and its applications. The *cost* of a path is the sum of the weights of its edges. Informed and uninformed search algorithms for finding *shortest* (minimal-cost) paths are heavily used in planning, scheduling, machine learning, constraint optimization, etc.

Graph edge-weights are commonly assumed to be available in negligible time. However, this does not hold in many applications. When weights are determined by queries to remote sources, or when a massive graph is stored in ex-

ternal memory (e.g., disk) then the order in which edges are visited—accessing external memory—needs to be optimized Vitter (2001); Hutchinson et al. (2003); Jabbar (2008); Korf (2008b,a, 2016); Sturtevant and Chen (2016). Similarly, when edge-weights are computed dynamically using learned models, or external procedures, it is beneficial to delay weight evaluation until necessary Dellin and Srinivasa (2016); Narayanan and Likhachev (2017); Mandalika et al. (2018, 2019).

Instead of delaying and re-ordering expensive edge-weight evaluations, we introduce here a formalization that focuses instead on using multiple *weight estimators*. Edge-weights are replaced with an ordered set of estimators, each providing lower and upper bounds on the true weight. Incrementally, subsequent estimators can tighten the bounds, but at increasing computation time. A search algorithm may quickly compute loose bounds on the edge-weight, and invest more computation on a tighter estimator later in the process.

For example, consider finding the fastest route between two cities, where edges and their weights represent roads and travel times, resp. First rough bounds on travel time can be estimated from fixed distances and speed limits (say, from a local database). For more accuracy, Google Maps, which considers additional road factors and traffic data, can be queried online. Even more accuracy can be achieved—at increased runtime—by further considering *vehicle attributes* together with road characteristics, which can be highly relevant for large and heavy vehicles such as trucks and buses PTV-Group (2023).

Having multiple weight estimators for edges is a proper generalization of standard edge-weights, and raises several shortest path problem variants. The classic singular edge-weight is a special case, of an estimator whose lower- and upper- bounds are identical. However, since the true weight may not be known (even applying the most expensive estimator), other variants of the shortest path problems involve finding paths that have the best bounds on the optimal cost.

This dissertation first introduces the *shortest path tightest lower-bound* (SLB) problem in Chapter 3, which is to find a path with the tightest lower bound on the optimal cost. SLB is an important shortest-path problem variant in graphs with multiple-estimated edge weights, since its solution provides a lower bound for

the true cost of *any* solution, even when costs are unknown, and furthermore it is key to determining optimal (or bounded-suboptimal) paths in such graphs, as we will discuss.

We present BEAUTY, an uninformed search algorithm based on *uniform-cost search* (UCS, a variant of *Dijksra's* algorithm) Felner (2011). We then use it to construct an anytime algorithm (A-BEAUTY) that provides additional flexibility for trade-offs between time spent on search and time spent on estimation. Both algorithms are shown to be correct and complete.

The dissertation continues to introduce the *tightest admissible shortest path* (TASP) problem in Chapter 4, which is a canonical shortest-path problem in graphs with multiple-estimated edge-weights. Its solution provides an answer to the question: what is the tightest suboptimality factor w.r.t. the optimal cost that can be achieved given that edge-weights have uncertainty bounds? We show that solving TASP can be reduced to solving two simpler problems: SLB, and the *shortest path tightest upper bound* (SUB) problem, which is defined and solved in Chapter 4.

To solve SUB, we present BEAST, an uninformed search algorithm based on *Uniform-Cost Search* (UCS, a variant of *Dijkstra's* algorithm) Dijkstra (1959); Felner (2011), that takes advantage of prior information about the solution of SUB. We then use it to construct BEAUTY&BEAST, an algorithm that solves TASP problems by exploiting coupling between the problems of SLB and SUB. Experiments with all the suggested algorithms demonstrate their effectiveness in the uninformed setting.

Finally, as shown in Section 2.5, heuristics can be defined and utilized for searching on EWDGs, and all our suggested algorithms for graph search can be immediately and naturally extended to work with heuristics. Thus, our results carry over to informed search. This is demonstrated in Chapter 6 where we perform planning on induced EWDGs with an informed search algorithm.

2.2 Motivating Example

One of the major sources of uncertainty in planning multi-segment flight routes for logistics drone missions is energy consumption (EC), which is affected by winds, weight, and other factors. Optimistic and pessimistic EC bounds can be very useful for better drone scheduling, e.g., by using the bounds to devise greedy/conservative schemes Jung and Kim (2022). Based on wind vector predictions Gianfelice et al. (2022) it is possible to estimate EC bounds in a flight segment (graph edge) in multiple ways Jung and Kim (2022); Gianfelice et al. (2022); Chan and Kam (2020). These may involve a mix of online queries and computation, and can be used to obtain the solutions to SLB, SUB, and TASP. Here the solution of TASP can be interpreted as the maximum energetic overhead for choosing a conservative route, or in case the tightest lower and upper bounds are both obtained for the same route, then the solution to TASP exactly quantifies its EC uncertainty. To sum up:

- EC uncertainty in drones is often significant, thus bounds on the EC are useful for mission planning.
- Estimation of EC bounds is possible by various methods. This is an active area of research.
- For each flight-segment (edge) we may use multiple methods, varying in computational cost.

Analogous examples exist for robots and unmanned vehicles, for a variety of measures of interest (such as travel time, pollution output and energy consumption) that depend on the medium traversed. There are many methods for estimating those, depending on the application, required precision, and available computational resources: from relatively simple engineering formulas, to physics-based simulators.

2.3 Background and Related Work

To put this research in context, consider the *abstract* components of the search runtime T, in a manner inspired by Mandalika et al. (2019):

$$T = T_w + T_v = \tau_w \times w + \tau_v \times v, \tag{2.1}$$

where T_w is the time spent on edge weight computation and T_v is the time spent on vertex (search) operations (e.g., expansion, queue operations, etc.). T_w, T_v can be decomposed as follows: w is the number of edge weight computations conducted, and v the number of vertices encountered during the search; τ_w, τ_v are respectively the average *edge weight computation time*, and average *vertex search operations* time (for every vertex considered).

We use this abstract view to examine different algorithmic approaches in terms of their efforts to reduce v or w, sometimes trading an increase in one parameter to reduce another. Standard search algorithms assume τ_w is negligible (or a small constant) and so their effort is mostly on reducing v. In contrast, algorithms for finding shortest paths in robot configuration spaces must consider settings where τ_w is *high*, since in these applications, edge existence and cost are determined by expensive computations for validating geometric and kinematic constraints. Thus these algorithms reduce w by explicitly delaying weight computations Dellin and Srinivasa (2016); Narayanan and Likhachev (2017); Mandalika et al. (2018, 2019), even at the cost of increasing v. Related challenges arise in planning, where action costs can be computed by external (lengthy) procedures Dornhege et al. (2012); Gregory et al. (2012b); Frances et al. (2017), or when multiple heuristics have different runtimes Karpas et al. (2018).

There are also approaches that seek to directly reduce τ_w (rather than w). When the graph is too large to fit in random-access memory, it is stored externally (i.e., disk). External-memory graph search algorithms optimize the memory access patterns for edges (and vertices), so as to make better use of faster memory (caching) Vitter (2001); Hutchinson et al. (2003); Jabbar (2008); Korf (2008b,a, 2016); Sturtevant and Chen (2016). This reduces τ_w by amortizing the computa-

tion costs, but still assumes a single weight computation per edge.

The approach we take here is complementary to those above. We consider a case where the weight of each edge can be estimated multiple times, successively, at increasing expense for greater accuracy. The component $T_w = \tau_w \times w$ is then replaced with

$$T_w = \tau_{w_1} \times w_1 + \tau_{w_2} \times w_2 \ldots \tau_{w_k} \times w_k, \tag{2.2}$$

with $\tau_{w_1} < \ldots < \tau_{w_k}$, and possibly, $w_1 + \ldots + w_k > w$. However, the total sum in Eq. 2.2 may be much smaller than the original $\tau_w \times w$. While naturally, $\tau_{w_k} \leq \tau_w$, the search algorithm may produce $w_k \ll w$. The algorithms presented here make use of this to balance search effort and edge evaluation in a refined manner, and thus to reduce overall runtime T_w.

This re-thinking of edge weights is independent of, and complementary to, other extensions to the definition of weights in graphs. For example, scalar weights can be *random*, drawn from a distribution associated with each edge Frank (1969). Fuzzy weights Okada and Gen (1994) allow quantification of uncertainty by grouping approximate weight ranges to several representative sets. Multi-objective weights Stewart and III (1991) allow each edge to be associated with a vector of different weights, facilitating optimization of multiple objectives. All of these extensions ignore the weight *computation time*, in contrast to the work reported here.

2.4 Shortest Path with Estimated Weights

Graph Definitions. A *weighted digraph* is a tuple (V, E, c), where V is a set of nodes, E is a set of edges, s.t. $e = (v_i, v_j) \in E$ iff there exists an edge from v_i to v_j, and $c : E \to \mathbb{R}^+$ is a cost (weight) function mapping each edge to a non-negative number. Let v_i and v_j be two nodes in V. A *path* $\pi = \langle e_1, \ldots, e_n \rangle$ from v_i to v_j is a sequence of edges $e_k = (v_{q_k}, v_{q_{k+1}})$ s.t. $k \in [1, n]$, $v_i = v_{q_1}$, and $v_j = v_{q_{n+1}}$. The cost of a path π is then defined to be $c(\pi) := \sum_{k=1}^{n} c(e_k)$. The *Goal-Directed Single-Source Shortest Path* (GDS^3P) problem involves finding a *solution*, which is a path π from the source node to a goal node, with minimal $c(\pi)$, denoted as C^*. A solution π for a GDS^3P problem is said to be a **$\mathcal{B}$-admissible shortest**

path if $c(\pi)$ is bounded by a suboptimality factor $\mathcal{B}$, i.e.,

$$c(\pi) \leq C^* \times \mathcal{B}. \tag{2.3}$$

If $\mathcal{B} = 1$, then π is a shortest path.

Definition 1 *A* **cost estimators function** Θ*, for a set of edges E and with a cost function $c : E \to \mathbb{R}^+$, maps every edge $e \in E$ to a finite and non-empty sequence of* weight estimation procedures*,*

$$\Theta(e) := (\theta_e^1, \ldots, \theta_e^{k(e)}), k(e) \in \mathbb{N}, \tag{2.4}$$

where **estimator** θ_e^i*, if applied, returns lower- and upper- bounds (l_e^i, u_e^i) on $c(e)$, such that $0 \leq l_e^i \leq c(e) \leq u_e^i < \infty$). $\Theta(e)$ is ordered by the increasing running time of θ_e^i, and the bounds monotonically tighten, i.e., $[l_e^j, u_e^j] \subseteq [l_e^i, u_e^i]$ for all $i < j$.*

Definition 2 *An* **estimated weighted digraph (EWDG)** *is a tuple $G = (V, E, c, \Theta)$, where V, 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.*

Definition 3 *For an edge e, the* **tightest edge lower bound** *and* **tightest edge upper bound** *w.r.t. Θ are $l_{\Theta(e)} := l_e^{k(e)}, u_{\Theta(e)} := u_e^{k(e)}$. For a path π, the* **tightest path lower bound** *and* **tightest path upper bound** *w.r.t. Θ follow, respectively, from the tightest edge bounds defined above.*

$$l_{\Theta(\pi)} := \sum_{i=1}^{n} l_{\Theta(e_i)}, \quad u_{\Theta(\pi)} := \sum_{i=1}^{n} u_{\Theta(e_i)} \tag{2.5}$$

Intuitively, $l_{\Theta(\pi)}$ and $u_{\Theta(\pi)}$ are the best estimations that are provided by Θ for the true cost of the path π, and they satisfy $l_{\Theta(\pi)} \leq c(\pi) \leq u_{\Theta(\pi)}$.

Shortest Path Problems. Regular weighted digraphs are a special case of EWDGs where for every edge e, there is a single estimation procedure $\theta_e^1 = (c(e), c(e))$ with lower and upper bounds that are identical to the weight $c(e)$. In this special case, a shortest tightly-bounded path π in the graph is an optimal solution for a GDS^3P problem. However, in the general case of EWDGs, multi-

ple estimators exist per edge, and we are not guaranteed that every weight can be estimated precisely, even if all estimators for it are used, as the best estimator available for an edge can still deviate from the exact true cost provided by c. Thus, several variants of the shortest path problem exist, which correspond to different tightest bounds for the shortest path.

A natural problem that is of interest here is that of finding the smallest possible suboptimality factor on the optimal cost C^* as well as finding a path π that achieves it, in a given EWDG. In order to properly define this problem, we first have to extend the notion of $\mathcal{B}$-admissibility to EWDGs. Indeed, Inequality (2.3), which characterizes the suboptimality factor, requires knowing exact edge costs to obtain $c(\pi)$ and C^*, but the cost function c is un-observable in EWDGs. Instead, we introduce an extension (Def. 4) that uses edge cost estimates that are provided by Θ. The extension relies on L^*, which is the best lower bound that can be derived for C^* (introduced in Weiss et al. (2023)):

$$L^* := \min_{\pi'}\{l_{\Theta(\pi')} \mid \pi' \text{ is a path from } v_s \text{ to } v \in V_g\}. \tag{2.6}$$

Definition 4 *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. A solution π is said to be a* **$\mathcal{B}$-admissible shortest path w.r.t.** *Θ if the following holds*

$$u_{\Theta(\pi)} \leq L^* \times \mathcal{B}. \tag{2.7}$$

Note that $u_{\Theta(\pi)}$ is the tightest upper bound for $c(\pi)$ w.r.t. Θ (Eq. (2.5)), so $c(\pi) \leq u_{\Theta(\pi)}$ holds. Similarly, L^* is the tightest lower bound for C^* w.r.t. Θ (Eq. (2.6)), so $L^* \leq C^*$ holds. Thus, if Inequality (2.7) holds, then

$$c(\pi) \leq u_{\Theta(\pi)} \leq L^* \times \mathcal{B} \leq C^* \times \mathcal{B} \tag{2.8}$$

is necessarily satisfied. Namely, standard $\mathcal{B}$-admissibility is assured by $\mathcal{B}$-admissibility w.r.t. Θ.

Next, we define a canonical problem addressed in this thesis (introduced in Weiss et al. (2024)).

Problem 1 (TASP, finding $\mathcal{B}^*$) *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 $\mathcal{B}^*$ be the tightest $\mathcal{B}$-admissibility factor w.r.t. Θ, i.e.,*

$$\mathcal{B}^* := \min_{\pi'}\{\mathcal{B} \mid u_{\Theta(\pi')} \leq L^* \times \mathcal{B}, \pi' \text{ is a solution}\}. \tag{2.9}$$

The **Tightest Admissible Shortest Path** *(TASP) problem is to find $\mathcal{B}^*$ as well as a solution π that its $\mathcal{B}$-admissibility factor is $\mathcal{B}^*$. If $\mathcal{B}$-admissibility cannot be obtained for any finite value of $\mathcal{B}$, or no solution exists, then $\mathcal{B}^* = \infty$ should be returned.*

Next, we describe two problems that are related to Prob. 1. The first problem among the two deals with finding a shortest path w.r.t. lower bounds (introduced in Weiss et al. (2023)).

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., $l(\pi) = L^*$.*

The second problem is complementary to Prob. 2 and deals with finding a shortest path w.r.t. upper bounds.

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 path upper bound of any path from v_s to $v \in V_g$, w.r.t. Θ, i.e., $u(\pi) = U^*$, with*

$$U^* := \min_{\pi'}\{u_{\Theta(\pi')} \mid \pi' \text{ is a path from } v_s \text{ to } v \in V_g\}. \tag{2.10}$$

We next show that Problems 1–3 are all generalizations of standard GDS^3P (see Thm. 1 below). In addition, they are also related to each other, in that their solutions are linked. Indeed, Thm. 2 below shows that the best lower bound for C^* (Prob. 2) and the best upper bound for C^* (Prob. 3) can be used to calculate the best suboptimality factor $\mathcal{B}^*$ (Prob. 1). Thm. 2 also shows that an optimal

solution path for Prob. 3 is also an optimal solution path for Prob. 1. The proof of Thm. 2 also implies that if $U^* > L^* = 0$ it is impossible to prove $\mathcal{B}$-admissibility at all.

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

Proof. *Any GDS^3P* problem can be formulated as Problem 1, or 2, or 3, by considering the special case where each edge has one estimator (namely, $k(e) = 1$ for every e), that returns the exact cost (i.e., $l_e^1 = c(e) = u_e^1$). In this special case it holds that $L^* = C^* = U^*$ and $\mathcal{B}^* = 1$ (in the case of $L^* = U^*$ we set $\mathcal{B}^* = 1$ even if $C^* = 0$ as there is no uncertainty at all). Therefore, the cost of the solutions of these problems are all equal to the minimum cost C^*, hence are by definition shortest paths. ■

Theorem 2 ($\mathcal{B}^* = U^*/L^*$) *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. 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 (i.e. a solution that achieves U^*). Furthermore, $\mathcal{B}^* = U^*/L^*$.*

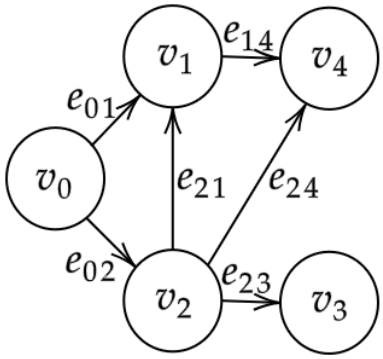
Proof. Assume $L^* > 0$ for P. By definition (of TASP and of $\mathcal{B}^*$) a solution π is a tightest admissible shortest path iff it achieves the lowest $\mathcal{B}$-admissibility factor, i.e., iff it satisfies

$$\pi = \arg\min_{\pi'} u_{\Theta(\pi')}/L^*. \tag{2.11}$$

Since L^* does not change with different choices of π' in the argmin expression of Eq. (2.11), it follows that

$$\pi = \arg\min_{\pi'} u_{\Theta(\pi')}. \tag{2.12}$$

By definition of U^* (Eq. (2.10)), a solution π satisfying Eq. (2.12) achieves $u_{\Theta(\pi)} = U^*$. On the other hand, a solution satisfying Eq. (2.10), also achieves $\mathcal{B}^*$, as implied by Eq. (2.11). Thus, π is a tightest admissible shortest path iff



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

Figure 2.1: Left: Digraph G. Right: costs and estimates of G.

it is a shortest path tightest upper bound (SUB). Furthermore, it holds that $\mathcal{B}^* = U^*/L^*$. ■

Corollary 1 *Thm. 2 shows how to obtain an optimal solution for Prob. 1 from optimal solutions for Problems 2 and 3. Thus, any algorithmic improvement to solving either Prob. 2 or Prob. 3 directly affects the efficiency of solving Prob. 1.*

Next, we provide an example (taken from Weiss et al. (2024)) to illustrate the meaning of the solutions for Problems 1–3.

Example 1 *Consider the estimated weighted digraph $G = (V, E, c, \Theta)$ provided in Fig. 2.1. Given the graph above, we may define the problem $P = (G, v_s, V_g)$ with $v_s = v_0$ and $V_g = \{v_3, v_4\}$, i.e., searching for paths from v_0 to either v_3, or v_4. Then, the unknown optimal cost is $C^* = c(\pi^*) = c_{01} + c_{14} = 9$ with $\pi^* = \langle e_{01}, e_{14} \rangle$; the tightest lower bound for C^* is $L^* = l_{\Theta(\pi_1)} = l^2_{02} + l^1_{24} = 7$ with $\pi_1 = \langle e_{02}, e_{24} \rangle$ (the SLB solution); the tightest upper bound for C^* is $U^* = u_{\Theta(\pi_1)} = u^1_{01} + u^2_{14} = 10$ with $\pi_2 = \pi^*$ (the SUB solution); and tightest admissibility factor is $\mathcal{B}^* = U^*/L^* = 10/7$ with $\pi_3 = \pi_2$ (the TASP solution).*

We remind the reader that Problem 2 (SLB) is solved in Chapter 3, and Problems 3 and 1 (SUB and TASP, correspondingly) are solved in Chapter 4.

Remark 1 *An assumption made in this section is that estimators represent cost uncertainty by an interval composed of deterministic lower and upper bounds. This supports the derivation of strong theoretical guaranties, motivating the assumption. However,*

deterministic bounds are not always available. In cases where statistical estimators are used instead, statistical properties (e.g., confidence intervals) may be used instead as bounds, in which case the plan accuracy bound should be considered soft (or approximate) rather than strict. Namely, the mathematical formulation introduced is still useful, but it no longer enables to derive some of the theoretical guaranties provided in this dissertation. A more elaborate discussion is given in Section 7.1.

2.5 Heuristics Based on Lower Bounds of Edge Costs

The basic idea underlying the optimality guaranty of informed search algorithms such as A* relies on using a heuristic function h with the property of consistency, which allows a best-first search based on $f = g + h$ values to be monotonically non-decreasing, and thus ensure that the first solution obtained is an optimal solution. Here, we show that a heuristic function that uses lower bounds of edge costs retains the theoretical property that is required to achieve path-cost optimality when used in a graph search.

Intuitively, a heuristic value that is calculated based on lower bounds of edge costs should only decrease relative to a heuristic value for the same vertex that is calculated with the same heuristic function but based on exact edge costs. This simple idea implies that heuristic admissibility is preserved. However, in order to address consistency a careful formal proof is necessary.

Note that often the term heuristic function, in the context of graph search (or AI planning), is an abuse of notation used to refer to a more general computational procedure, that is parameterized by the edge costs of the graph (or by the action costs in the planning problem), so that only after fixing the costs, a standard heuristic is obtained. Further note that it is typical to attribute a theoretical property, such as consistency, to such a computational procedure, if the property holds for any heuristic obtained from the procedure after fixing the costs, regardless of their specific values. For stating our result, we need to differentiate between the two. Hence, we call the more general computational procedure a parameterized heuristic and denote it as usual by h, and we denote

a standard heuristic obtained by it using a subscript, e.g., h_c, where c is a cost function defining the costs of the problem.

Lemma 1 (Consistency Preservation) *Given a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and two edge cost functions $\alpha : \mathcal{E} \to [0, \infty)$, $\beta : \mathcal{E} \to [0, \infty)$ that satisfy $\alpha(e) \leq \beta(e)$ for every edge $e \in \mathcal{E}$, we obtain the weighted digraphs $\mathcal{G}_\alpha = (\mathcal{V}, \mathcal{E}, \alpha)$, $\mathcal{G}_\beta = (\mathcal{V}, \mathcal{E}, \beta)$. If h is a consistent parameterized heuristic, then h_α is consistent w.r.t. $\mathcal{G}_\beta$.*

Proof. Due to the fact that h is a consistent parameterized heuristic, it immediately follows that h_α is consistent w.r.t. $\mathcal{G}_\alpha$. From the definition of consistency, this means that $h_\alpha(n) \leq \alpha((n,s)) + h_\alpha(s)$ and $h_\alpha(G) = 0$ is satisfied for every node n and every descendant s of n, and every goal node G in $\mathcal{G}_\alpha$. Using $\alpha(e) \leq \beta(e)$ we obtain $h_\alpha(n) \leq \beta((n,s)) + h_\alpha(s)$, where again, this holds for every node n and every descendant s of n. Additionally, every goal node G in $\mathcal{G}_\alpha$ is also a goal node in $\mathcal{G}_\beta$, so $h_\alpha(G) = 0$ is also satisfied for every goal node in $\mathcal{G}_\beta$. Thus, h_α satisfies the definition of consistency w.r.t. $\mathcal{G}_\beta$. ■

Chapter 3

Optimally Solving Shortest Path Tightest Lower-Bound

"An approximate answer to the right question is worth a great deal more than a precise answer to the wrong question."

John Wilder Tukey

Based on:
Eyal Weiss, Ariel Felner, Gal A. Kaminka. "A Generalization of the Shortest Path Problem to Graphs with Multiple Edge-Cost Estimates" In *Proceedings of the European Conference on Artificial Intelligence: ECAI 2023*, pp. 2607-2614. 2023.

3.1 Algorithms for SLB

We present two algorithms for solving the SLB problem. Both aim at reducing the number of expensive estimators used. The first algorithm, BEAUTY (Branch&bound Estimation Applied in UCS To Yield bottom, Alg. 1), extends UCS to dynamically apply cost estimators during a best-first search w.r.t. lower bounds of edge costs. The second algorithm, A-BEAUTY (Anytime BEAUTY, Alg. 3) uses BEAUTY in iterations, such that bounds established in one iteration

are used to focus the search in the next, monotonically improving the solution. Both algorithms are proved correct and complete.

3.1.1 The First Algorithm: BEAUTY

Algorithm 1 receives an SLB problem instance and two hyper-parameters l_{est}, l_{prune}. For simplicity we will first describe a *base case* where l_{est}, l_{prune} are both set to ∞, and therefore have no effect and can be ignored. The relevant lines using l_{est} and l_{prune} are colored in blue (Lines 17–19) and should be ignored for now. We will come back to these hyper-parameters later to provide full explanations for how they are used and what their purpose is, but for now we provide a simple intuition: they can be used to encode prior information on L^* that makes the search more efficient.

Algorithm 1 BEAUTY

Input: Problem $P = (G, \Theta, v_s, V_g)$
Parameter: Thresholds l_{est}, l_{prune}
Output: Path π, Opt, bounds $\underline{l}^*, \bar{l}^*$

$g_l(s_0) \leftarrow 0$; OPEN $\leftarrow \emptyset$; CLOSED $\leftarrow \emptyset$
Insert s_0 into OPEN with $g_l(s_0)$
while OPEN $\neq \emptyset$ **do**
  $n \leftarrow$ pop node n from OPEN with minimal $g_l(n)$
  if $Goal(n)$ **then**
    $l(\pi) \leftarrow g_l(n)$
    $Opt, \underline{l}^*, \bar{l}^* \leftarrow$ BEAUTY-PS
    return $trace(n), Opt, \underline{l}^*, \bar{l}^*$
  Insert n into CLOSED
  for each successor s of n **do**
    if s not in OPEN $\cup$ CLOSED **then**
      $g_l(s) \leftarrow \infty$
    $\tilde{g}_l \leftarrow g_l(n)$
    while $\tilde{g}_l < g_l(s)$ **and** estimators remain for $e = (n, s)$ **do**
      $l(e) \leftarrow$ Apply next estimator for e
      $\tilde{g}_l \leftarrow g_l(n) + l(e)$
      if $\tilde{g}_l > l_{est}$ **then**
        break // exit while loop
    if $\tilde{g}_l < g_l(s)$ **and** $\tilde{g}_l \leq l_{prune}$ **then**
      $g_l(s) \leftarrow \tilde{g}_l$
      if s in OPEN **then**
        Remove s from OPEN
      Insert s into OPEN with $g_l(s)$
return $\emptyset, false, \infty, \infty$

Procedure 2 BEAUTY-PS
Input: BEAUTY's inputs and variables
Output: Opt, bounds $\underline{l}^*, \bar{l}^*$

$Opt \leftarrow true$; $\underline{l}^* \leftarrow l(\pi)$
for each edge e in π **do**
  if estimators remain for e **then**
    $l \leftarrow$ Apply the best estimator for e
    $l(\pi) \leftarrow l(\pi) + l - l(e)$
    $l(e) \leftarrow l$
if $l(\pi) > \underline{l}^*$ **then**
  $Opt \leftarrow false$
$\bar{l}^* \leftarrow l(\pi)$
return Opt, $\underline{l}^*$, $\bar{l}^*$

Base Setting. BEAUTY is structurally similar to UCS. It activates a best-first search process using the standard OPEN and CLOSED lists. Nodes n in OPEN are prioritized by $g_l(n)$ which is, in the base case, always equal to the optimal lower bound to node n along the best known path (similar to using $g(n)$ for ordering nodes in UCS in regular graphs, which is done according to optimal cost). The *best* such node n is chosen for expansion in Line 4, and its successors are added in the loop of Lines 10–23. The main change of BEAUTY over UCS is in the *duplicate detection* mechanism performed when evaluating the cost of a new edge e that connects n to its successor s. In UCS, the exact edge cost $c(e)$ is immediately obtained and used to update the path cost that ends in s. In BEAUTY, we iterate over the different estimators θ_e^i for edge e (Lines 14–16). In each iteration we set $\tilde{g}_l$ to be the lower bound for the path to s given the current estimator (Line 16). Now such a path can be already pruned earlier if its current lower bound (using the current estimator) will not improve the best known path to s ($g_l(s)$). In that case we will not need to activate the entire set of estimators (in particular, the expensive ones). Thus, if $\tilde{g}_l \geq g_l(s)$, the while statement (Line 14) ends. Then, ordinary duplicate detection is performed in Lines 19–23. See Example 2 for a demonstration of using BEAUTY in its base setting.

Enhanced Setting. We now consider the enhanced setting where l_{est}, l_{prune} are set to some constant values (not ∞). First, l_{est} is used as an upper bound for

activating the series of estimators. When a node n has a path lower bound $> l_{est}$ then we no longer activate the series of estimators and only apply the first (cheapest) estimator for edges after n (including the edge to n). This is done in Lines 17–18 where we break the loop that further activates estimators on the current edge. Second, l_{prune} is used as an upper bound to prune (and not add to OPEN) any node with lower bound $> l_{prune}$ (in a similar manner to *Bounded Cost Search* Stern et al. (2014)). This is done in Line 19.

The purpose of using $L^* \leq l_{est} < \infty$ is to avoid applications of redundant (and expensive) estimators. Similarly, the purpose of using $L^* \leq l_{prune} < \infty$ is to decrease the size of OPEN, which implies less insertion operations and cheaper insert/delete operations. But since L^* is unknown, setting these hyper-parameters to meaningful values requires prior information. Practically, such information can be achieved by obtaining a suboptimal solution with $l \geq L^*$, and using it to set $l_{est} = l_{prune} = l$. This idea is implemented in the anytime algorithm (A-BEAUTY) discussed below.

Goal Test and the Post-Search Procedure BEAUTY-PS**.** When a solution π is found by the *Goal* function (Line 5), with the path lower bound $l(\pi)$, BEAUTY calls BEAUTY-PS (post-search procedure, Proc. 2 below) to iterate over the edges of π and tighten the estimations whenever possible, to produce the tightest lower bound $\bar{l}^*$ for π. If $\bar{l}^* = l(\pi)$, namely the path bounds were already tight before BEAUTY-PS, then it determines that π is optimal and sets $Opt \leftarrow true$. Note that in the base setting when a solution is found it is always already tightly estimated before BEAUTY-PS, so no further estimators are applied and $Opt \leftarrow true$. BEAUTY-PS returns $Opt, \underline{l}^* = l(\pi)$ and $\bar{l}^*$, which are then returned by BEAUTY together with π (generated by a path-reconstruction function *trace*).

Depending on the hyper-parameters l_{prune}, l_{est}, BEAUTY is *complete* (Lemma 2), *sound* (Lemma 3), and *optimal* (Lemma 4).

Lemma 2 (Conditional Completeness Prob. 2) BEAUTY*, provided with* $l_{prune} \geq L^*$*, is complete.*

Proof. BEAUTY inspects nodes that are removed from OPEN by best-first order w.r.t. lower bound of path cost. When $l_{prune} = \infty$ is satisfied, no node is pruned, so that every node encountered during the search is inserted into OPEN. The condition $\tilde{g}_l < g_l(s)$ simply verifies that each node in OPEN points back to the best found path leading to it, but it does not prevent nodes from being inserted. In this case completeness is assured, as the search is systematic.

Suppose that a best-first algorithm utilizes all possible estimators per edge it encounters. Then, if a solution exists, a shortest path tightest lower bound π^* will necessarily be returned with L^*. Since applying more estimators can only increase (tighten) the lower bound for an edge, it follows that when not all possible estimators per edge are utilized, and a systematic best-first search takes place, then a solution π for P ending in a node n will be found, where the key of n in OPEN (the obtained lower bound), immediately before it was removed, must be lower than, or equal to, L^*. This holds regardless of the value of l_{est}, that only affects which (and how many) estimators will be applied. Namely, the value of l_{est} may affect which solution π is found, but not the fact that such a solution will be found. Hence, when $l_{prune} \geq L^*$ is satisfied, a solution π is necessarily found. ■

Lemma 3 (Bounds for L^*) BEAUTY, *provided with $l_{prune} \geq L^*$, returns $0 \leq \underline{l}^* \leq L^* \leq \bar{l}^*$, if a solution exists for P. Furthermore, if $l_{est} < L^*$ also holds, then $\underline{l}^* > l_{est}$.*

Proof. The proof of Lemma 2 established that when BEAUTY is called with $l_{prune} \geq L^*$, a solution π will be found (when a solution exists), ending in a node n, where the key of n in OPEN $g_l(n)$ (the obtained lower bound), immediately before it was removed, satisfies $g_l(n) \leq L^*$. Additionally, $g_l(n) \geq 0$ trivially holds, as each edge lower bound is by definition non-negative. In line 6 of BEAUTY $l(\pi) \leftarrow g_l(n)$ is set, then BEAUTY-PS is called, which sets $\underline{l}^* \leftarrow l(\pi)$ in Line 1, and then $\underline{l}^*$ is not changed until it is returned. BEAUTY-PS utilizes all unused estimators in the solution π, by systematically improving estimations for each edge e belonging to π using all estimators in $\Theta(e)$. Thus the tightest possible lower bound for π is obtained and returned as $\bar{l}^*$. From the optimality of L^* it follows that $\bar{l}^* \geq L^*$. To sum up, $\underline{l}^*, \bar{l}^*$, that satisfy $0 \leq \underline{l}^* \leq L^* \leq \bar{l}^*$, are

returned.

Let us now consider the case that $l_{est} < L^*$ holds in addition to $l_{prune} \geq L^*$. Seeking a contradiction, assume that $\underline{l}^* > l_{est}$ is not necessarily satisfied. This means that for some solution π, it holds that $\underline{l}^* \leq l_{est}$. Recall that $\underline{l}^* = g_l(n)$ for the node n, which is the last node in the path implied by the solution π. Since $l_{est} < L^*$ holds, it must be that each edge in π has been estimated using all possible estimators before n is established as a goal node, as for each node n' satisfying the condition $g_l(n') \leq l_{est}$, edges included in the path leading to n' are only denied tight estimation in cases where a better alternative path leading to n' was already found. Therefore, the lower bound of π cannot be tightened, so $\underline{l}^* = \bar{l}^*$ is satisfied, implying that π is optimal with lower bound L^*. But this means that $L^* = \underline{l}^* \leq l_{est} < L^*$. A contradiction. Hence, $\underline{l}^* > l_{est}$. ■

Lemma 4 (Conditional Optimality Prob. 2) BEAUTY, *provided with* $l_{prune} \geq L^*$ *and* $l_{est} \geq L^*$, *returns a shortest path tightest lower bound* π *and* $\bar{l}^* = L^*$, *if a solution exists for* P.

Proof. Continuing the argument made in the proof of Lemma 3, if $l_{prune} \geq L^*$ and $l_{est} \geq L^*$ hold, then the best paths, based on tightest possible estimates, with cumulative lower bounds of up to l_{est} are found, and their terminal nodes are inserted to OPEN. In particular, the best paths up to L^* (including this value) are found. From the definition of L^* it follows that there exists a solution π with a tight lower bound equal to L^*. Hence, π, or possibly another solution with the same tight lower bound, is guaranteed to be found when its corresponding goal node is removed from OPEN. Then, $\bar{l}^* = \underline{l}^* = L^*$ together with π are returned. ■

The implication of Lemmas 2–4 is that *SLB problems can be solved optimally using* BEAUTY by setting l_{prune} and l_{est} to be greater than, or equal to, L^*, which can always be achieved by setting them to ∞, as Example 2 shows.

Example 2 *Consider calling* BEAUTY *with* $l_{est} = l_{prune} = \infty$ *(i.e., base setting) on* P *from Example 1. Tracing its run, at the first iteration of the while loop it invokes* $\theta^1_{e_{01}}, \theta^1_{e_{02}}$ *and* $\theta^2_{e_{02}}$ *and inserts* v_1, v_2 *to OPEN with keys* $4, 3$. *At the second iteration*

Algorithm 3 A-BEAUTY

Input: Problem $P = (G, \Theta, v_s, V_g)$
Output: Path π, bound $\bar{l}^*$

```
l*_ ← 0; l̄* ← ∞; Opt ← false
while not Opt do
    π, Opt, l*_, l̄ ← BEAUTY (P, l*_, l̄*)
    if π = ∅ then
        return ∅, ∞
    if l̄ < l̄* then
        l̄* ← l̄
    Print π, l*_, l̄*
return π, l̄*
```

v_2 is removed from OPEN, $\theta^1_{e_{21}}, \theta^1_{e_{23}}, \theta^2_{e_{23}}, \theta^1_{e_{24}}$ are invoked, and v_3, v_4 are inserted to OPEN with keys $10, 7$. *At the third iteration v_1 is removed from OPEN, $\theta^1_{e_{14}}$ and $\theta^2_{e_{14}}$ are invoked. At the forth iteration v_4 is removed from OPEN and* BEAUTY *returns* $\langle e_{02}, e_{24} \rangle, true, 7, 7$.

However, a lower value of l_{est} enables to avoid redundant estimations, where the potential savings grow as l_{est} approaches L^* from above. This motivates the use of BEAUTY in an iterative framework that gradually increases l_{est} until the optimal solution is found.

3.1.2 The Second Algorithm: Anytime BEAUTY

The A-BEAUTY algorithm automates the iterative usage of BEAUTY with increasingly tightened l_{est} and l_{prune} around L^*, until the optimal solution is found. It starts with $l_{est} = 0$ and $l_{prune} = \infty$, and each time BEAUTY terminates it returns $\underline{l}^* > l_{est}$ (Lemma 3), which is used as l_{est} in the next call. Similarly, the returned $\bar{l}^*$ is a finite value (when a solution exists) that always is greater than, or equal to, L^* (again, Lemma 3). Using the lowest value of $\bar{l}^*$, l_{prune} is monotonically non-increasing.

The process converges in a finite number of iterations (shown below) and thus assures optimality, while gradually utilizing more estimations, that in turn support better approximations for L^* (which are saved every time an improvement

is achieved). The estimations are saved between iterations, so that it is not necessary to re-apply estimators. Technically, this is obtained by defining the next estimator to apply to first look for a saved value and only then turn to unused estimators. Tightened l_{prune} values decrease the size of OPEN, reducing memory consumption and runtime (due to less insertion operations, and cheaper insert/delete operations).

Theorem 3 (Completeness, Soundness and Optimality Prob. 2) A-BEAUTY *is complete. If a solution exists for P, then a shortest path tightest lower bound π and L^* are returned.*

Proof. A-BEAUTY initializes $\underline{l}^* \leftarrow 0$ and $\bar{l}^* \leftarrow \infty$, and then enters a loop that terminates when no solution is found or when the optimal solution is found. At each iteration of the loop, it calls BEAUTY with $l_{est} = \underline{l}^*$ and $l_{prune} = \bar{l}^*$. Due to the initialization, the conditions of Lemmas 2 and 3 are fulfilled in the first iteration, so that if a solution exists, a solution would be returned by BEAUTY, with tightened bounds, i.e., $\underline{l}^* > 0$ and $L^* \leq \bar{l}^* < \infty$. In the second iteration (if the optimal solution has yet to be found) the $\underline{l}^*$ and $\bar{l}^*$ found in the first iteration are used again as $l_{est} = \underline{l}^*$ and $l_{prune} = \bar{l}^*$ in the call for BEAUTY, where again the conditions for both lemmas hold. Thus $\underline{l}^*$ is guaranteed to monotonically increase with each iteration, and $\bar{l}^*$ can either decrease (but remain at least L^*) or stay the same. Hence, the conditions for both lemmas are satisfied in every iteration until termination, i.e., we have established that the conditional completeness of BEAUTY implies regular completeness for A-BEAUTY, and that $\bar{l}^*$ monotonically non-increases.

To show optimality, we next analyze the increase in $\underline{l}^*$ between subsequent iterations. Denote $\delta_i := \underline{l}^*_i - \underline{l}^*_{i-1}$, where $\underline{l}^*_i$ is the value obtained after call i to BEAUTY. Note that δ_i cannot be arbitrarily small values, as they exactly represent the differences between cumulative lower bounds of solutions obtained in subsequent iterations, which are limited to a finite set of values (induced by Θ). Thus, there exists a constant $\delta_{min} > 0$ such $\forall i, \delta_i \geq \delta_{min}$ is satisfied. Hence, either the optimal solution is found before $\underline{l}^*$ reaches L^*, or it is found right after it reaches it (Lemma 4), which necessarily occurs after a finite number of

iterations. ■

The proof of Thm. 3 shows the number of iterations until convergence to optimality is unknown a-priori. Nevertheless, we can set a simple threshold either on the number of iterations or on the convergence implied by $\bar{l}^*/\underline{l}^*$. Once the threshold is crossed, setting both l_{est} and l_{prune} to $\bar{l}^*$ ensures the last iteration. See Example 3.

Example 3 *Consider again the SLB problem P from Example 1. When calling* A-BEAUTY *on P, at the first iteration the utilized estimators are $\theta^1_{e_{01}}, \theta^1_{e_{02}}, \theta^1_{e_{14}}, \theta^2_{e_{14}}, \theta^1_{e_{21}}, \theta^1_{e_{23}}$ and $\theta^1_{e_{24}}$, where $\theta^2_{e_{14}}$ is invoked by* BEAUTY-PS. *The algorithm prints $\langle e_{01}, e_{14}\rangle, 5, 8$. At the second iteration the estimator $\theta^2_{e_{02}}$ is also utilized. The algorithm prints $\langle e_{02}, e_{24}\rangle, 7, 7$ and returns $\langle e_{02}, e_{24}\rangle, 7$.*

3.2 Empirical Evaluation for SLB

The theoretical guarantees of BEAUTY and A-BEAUTY touch on their optimality and completeness, but do not provide information as to the runtime savings they offer. We therefore empirically evaluate the algorithms in diverse settings, based on AI planning benchmark problems that were modified to have multiple action-cost estimators, so that these induce SLB problems.

The set of problems was taken from a collection of IPC (International Planning Competition) benchmark instances[1]. Starting from the full collection, we first filtered out every domain that didn't offer support for action costs. Then, for some of the domains we created additional problems by using different configurations of costs. For all problems and domains, we synthesized three estimators. Each edge e with cost $c_{old}(e)$ was mapped to a new cost $c_{new}(e)$ that satisfies $c_{new}(e) \geq c_{old}(e) \times f_3$, with $f_3 > f_2 > f_1 \geq 1$, so that $l^1_e := c_{old} \times f_1, l^2_e := c_{old} \times f_2, l^3_e := c_{old} \times f_3$ served as its first, second and third lower bound estimates. To diversify the estimator sets for different edges, the parameters f_1, f_2, f_3 were taken from the sets $f_1 \in \{1,2,3\}, f_2 \in \{f_1+1, f_1+2, f_1+3\}, f_3 \in \{f_2+1\}$, which resulted in nine different configurations. The choice of configuration was

[1] See `https://github.com/aibasel/downward-benchmarks`.

Table 1: The configuration of f_1, f_2, f_3 in Rows 2–4 according to the hash values displayed in Row 1.

Hash	1	2	3	4	5	6	7	8	9
f_1	1	2	3	1	2	3	1	2	3
f_2	2	3	4	3	4	5	4	5	6
f_3	3	4	5	4	5	6	5	6	7

taken according to the result of a simple hash function, that depends on $c_{old}(e)$ and a user-input seed, described as follows:

$$\text{Hash} = (c_{old}(e) + \text{seed}) \mod 9. \tag{3.1}$$

Then, the configuration was set according to Table 1. Each problem was run once per seed, where the seeds where taken from the set $[0, 8]$, which resulted in 9 instances per problem. Overall, this resulted in a cumulative set of 914 problem instances, spanning 12 unique domains. The full list of the domains and problems that were used in the experiments is detailed in Weiss and Kaminka (2023).

We note that the configurations depicted in Table 1 that are chosen according to the hash function of Eq. 3.1 guarantee that the same ground action, in different states, will have the same cost estimates.

BEAUTY and A-BEAUTY were implemented as search algorithms in *PlanDEM* (Planning with Dynamically Estimated Action Models (see Weiss and Kaminka (2023)). a C++ planner that extends Fast Downward (FD) Helmert (2006) (v20.06). All experiments were run on an Intel i7-1165G7 CPU (2.8GHz), with 32GB of RAM, in Linux. We also implemented *Estimation-time Indifferent UCS* (EI-UCS), a UCS algorithm that uses the most accurate estimate on each edge it encounters, to serve as a baseline. For every problem instance we ran EI-UCS, BEAUTY with $l_{est} = l_{prune} = \infty$, and two versions of A-BEAUTY—A-BEAUTY-2 and A-BEAUTY-10—with maximal number of 2 and 10 iterations, resp. We emphasize that all these algorithms are guaranteed to achieve optimal solutions. We report the results from problem instances which all algorithms solved successfully, i.e., found optimal solutions, within 5 minutes.

3.2.1 BEAUTY **vs.** EI-UCS

We begin by contrasting BEAUTY and EI-UCS, to examine the effectiveness of BEAUTY in avoiding unnecessary expensive estimations. BEAUTY is only guaranteed optimal if its two hyper-parameters, l_{est}, l_{prune}, are greater than L^*, which is unknown a-priori. Thus, to ensure a fair comparison, we set $l_{est} = l_{prune} = \infty$ for all the runs of BEAUTY (that are not part of the anytime framework). Using these settings, the only difference between BEAUTY and EI-UCS is the condition $\tilde{g}_l < g_l(s)$ in the estimation loop (line 15 in Alg. 1) that prevents applying further estimators when an alternative path with lower g-value is already known. In contrast, EI-UCS ignores estimator time, always computing the tightest lower bound possible for every edge. Hence, the two algorithms follow the exact same search mechanism (i.e., identical node expansion order), and may only differ in the numbers and types of the estimators applied. Of specific interest is the difference in *expensive* third-layer estimators usage. Note that under this setting BEAUTY-PS has nothing to improve, as the solution path is already fully estimated.

We denote by L_3 the number of third-layer estimators applied during search. The results are summarized below:

- The ratio $r_{L_3} := L_3(\text{BEAUTY})/L_3(\text{EI-UCS})$ had average of 60.82% (stddev 11.57%), median 60.88%, with overall range spanning 24.4% to 88.48%.
- Whenever BEAUTY did not apply a third-layer estimator for an edge, it used on average a second-layer estimator in 0.51% of the cases. Namely, in these cases estimation time was almost always dramatically reduced.

Table 2 reports the results for all algorithms, compared to EI-UCS. The results are grouped by domain (domains listed by row—see caption for column explanation). The table shows (third column, total for all domains in the last row) that roughly 40% (100-60.82) of the expensive estimations are avoided, on average. There is high variance, whose causes remain unknown for now.

Table 2: Summarized performance data of BEAUTY (∞, ∞), A-BEAUTY-2 and A-BEAUTY-10 (written as Any-2 and Any-10 for brevity) relative to EI-UCS, with breakdown by domains. For each algorithm and domain two entries are presented with average $\pm$ standard deviation in percentage: the ratio of third-layer estimator usage r_{L_3}(Alg):= L_3(Alg)/L_3(EI-UCS) and the ratio of expanded nodes $r_{\exp}$(Alg):=expanded(Alg)/expanded(EI-UCS).

Domain	#Instances	r_{L_3}(BEAUTY)	$r_{\exp}$(BEAUTY)	r_{L_3}(Any-2)	$r_{\exp}$(Any-2)	r_{L_3}(Any-10)	$r_{\exp}$(Any-10)
Barman	495	58.23±4.52	100±0	49.27±13.29	189.1±20.13	48.91±13.39	837.32±136.03
Caldera	72	83.25±3.01	100±0	58.48±8.08	176.42±9.72	57.88±8.42	905.15±90.99
Cavediving	54	70.77±0.78	100±0	59.26±3.33	200±0	59.26±3.33	981.48±47.88
Elevators	27	28.81±3.23	100±0	10.13±6.9	145.45±27.48	6.4±5.18	724.26±210.48
Floortile	36	54.83±0.76	100±0	45.13±7.51	183.53±13.31	44.6±7.68	890.43±100.06
Parcprinter	36	83.12±2.62	100±0	25.02±11.24	136.34±15.58	22.38±9.93	810.26±98.03
Scanalyzer	18	48.18±1.65	100±0	48.16±1.66	200±0	48.16±1.66	994.44±23.57
Settlers	36	71.87±2.22	100±0	40.88±13.61	177.87±20.82	35.34±15.16	692.36±142.32
Sokoban	36	52.24±0.9	100±0	49.2±2.44	196.34±4.2	48.89±2.68	934.51±94.83
Tetris	45	63.3±4.74	100±0	41.91±7.27	180.3±10.41	41.09±8.37	907.11±126.24
Transport	41	47.25±4.09	100±0	17.53±8.76	144.92±20.83	16.01±8.73	760.46±132.08
Woodworking	18	61.39±1.54	100±0	44.35±6.09	185.21±8.7	37.95±6.33	816±183.54
All domains	**914**	**60.82±11.57**	**100±0**	**46.03±15.75**	**182.67±23.66**	**45.13±16.37**	**849.65±142.31**

3.2.2 A-BEAUTY vs others

We now turn to discuss A-BEAUTY-2 and A-BEAUTY-10. The relevant experiment results are summarized in Columns 5, 6 (A-BEAUTY-2) and 7, 8 (A-BEAUTY-10) of Table 2.

First and foremost, the results reveal that A-BEAUTY-2 and A-BEAUTY-10 save roughly 54% (100-46) and 55% (100-45) of the most expensive estimations, compared to EI-UCS. This represents an additional 15% savings on top of BEAUTY.

Second, although both have relatively high standard deviations (about 16%), they perform similarly in most domains (see below for the exception). This can be attributed to the (typically) very informed upper bound $\bar{l}^*$ that is achieved after the first iteration, so there is little room for improvement. Indeed, the lower bound $\underline{l}^*$ typically comes very close to L^* when A-BEAUTY-10 converges, so when l_{est} is set to $\bar{l}^*$ after the first iteration of A-BEAUTY-2, it achieves an almost identical behavior as in the last iteration of A-BEAUTY-10.

We examined more closely the domains where the savings of A-BEAUTY-2 and A-BEAUTY-10 vary noticeably (e.g., in the Elevators domain). We observed that in many of these problems, the range of values for c_{old}, and thus also the range

of values for the lower bound estimates (induced by c_{old}), is relatively high compared to other domains, i.e., the interval $[A, B] \subset [0, \infty)$ from which the values are taken is relatively large. This implies a less smooth distribution of costs (and estimates) over the graph edges, where it is common to have significant jumps in g-values between two subsequent nodes on a path. The implication of such jumps is that it becomes easier to avoid estimation of non-relevant paths (with $g_l > l_{est}$). In the same cases of larger ranges of values, A-BEAUTY-10 more frequently achieves improved estimation savings compared to A-BEAUTY-2. We believe this may be due to the distribution of costs being less smooth, decreasing the likelihood that $\bar{l}^*$ ends up close to L^* after the first iteration, and allowing more room for improvement in additional iterations.

Finally, Table 2 shows that the two algorithms consume on average roughly 1.8 and 8.5 times the number of expanded nodes of EI-UCS, which is due to the search restart at every iteration. In domains where the estimation savings are similar, it appears that two iterations may be sufficient, and will be much more efficient. However, more generally—and recalling the abstracted runtime from earlier—this is a good example of how algorithms may increase the search operations, to save on weight computations. For instance, if the times spent on estimation and search (T_w and T_v resp., Eq. 2.1) satisfy $T_w = 10 \times T_v$ for EI-UCS and some problem P, then considering a typical factor two of savings in estimation time and twice the search time of A-BEAUTY-2 on P, it follows that the latter achieves overall runtime $T_2 = 0.5 \times T_w + 2 \times T_v = 7 \times T_v$ vs. $T_1 = T_w + T_v = 11 \times T_v$, i.e., a reduction of $\approx 36\%$ in runtime.

Table 3 provides additional information that sheds light on the development of search and estimation metrics throughout the iterations. The table follows the iterations of A-BEAUTY-10. Row 2 indicates the number of times convergence to an optimal solution occurred at iteration i, allowing us to examine how many iterations were needed to solve the problems, on average. As can be seen, 50% of the problems take less than 10 iterations, with rapid decrease from $i = 9$ down to $i = 4$, while the other 50% terminate at $i = 10$ or more (the maximum number of iterations in these experiments was 10). Row 3 reveals the convergence of the lower bound obtained to the terminal value L^*. We can see that

Table 3: Convergence analysis of A-BEAUTY-10. Row 2 indicates the number of times convergence occurred at iteration i, Rows 3 and 4 indicate the mean μ and standard deviation σ, respectively, for the ratio of the lower bound obtained after iteration i to L^*, where the values in Rows 2–4 are in percentages. Results are rounded to integers for ease of presentation.

Iteration i	1	2	3	4	5	6	7	8	9
Final i(%)	0	0	0	0	1	3	10	16	20
$\mu(\underline{l}_i^*/L^*)$(%)	40	63	76	85	90	94	95	97	97
$\sigma(\underline{l}_i^*/L^*)$(%)	7	9	9	8	7	6	5	4	4

Table 4: Pruning analysis of A-BEAUTY-10. Rows 2 and 3 indicate the mean μ and standard deviation σ, respectively, for the ratio between pruned nodes and evaluated nodes, in percentages. Results are rounded to integers.

Iteration i	1	2	3	4	5	6	7	8	9	10
μ(pr/ev)(%)	0	1	2	4	10	11	12	15	17	26
σ(pr/ev)(%)	0	5	7	10	16	16	17	18	19	22

the rate of convergence is decaying. Row 4 further strengthen this observation, as the standard deviations are relatively low and also decaying. This motivates using a maximum threshold to avoid a very long convergence process, which could incur significant search effort overhead.

Lastly, Table 4 shows the average and standard deviation (Rows 2 and 3, respectively) of pruned nodes out of evaluated nodes, for each iteration of A-BEAUTY-10, in percentage. It can be seen that the average percentage of pruned nodes is monotonically non-decreasing with the iterations, from roughly 1% at the second iteration to 26% at the tenth iteration, which is due to the monotonically non-decreasing upper bound l_{prune}, that serves for pruning. Namely, as the upper bound gets tighter, pruning becomes more effective.

3.2.3 BEAUTY-PS

Given that often, two iterations of A-BEAUTY offered the same savings as ten iterations, yet significantly more than a single iteration, it is interesting to examine the role of BEAUTY-PS (Procedure 2) in improving the results from the first iteration of A-BEAUTY. Recall that BEAUTY-PS obtains the tightest possible

lower bound $\bar{l}^*$ for $c(\pi)$, which can then either be interpreted as L^* if $opt = true$ is returned, or as an upper bound for L^* otherwise. When BEAUTY is called with its hyper-parameters set to ∞, it is optimal; BEAUTY-PS has nothing to improve. However, when it is called as part of A-BEAUTY, the hyper-parameters are different, which gives BEAUTY-PS the potential to improve the results before the next iteration.

The results provide insight to the efficacy of this procedure. When calling BEAUTY-PS after BEAUTY is run with $l_{est} = 0$ and $l_{prune} = \infty$ (the least informative hyper-parameters), BEAUTY-PS returns on average $\bar{l}^* = 1.0082 \times L^*$, i.e., only 0.82% higher than L^*, with standard deviation of 3.31%, where in the worst case $\bar{l}^*$ was 33.33% higher than L^*. This means that just one iteration of BEAUTY that uses the cheapest lower bounds during the search, followed by BEAUTY-PS, typically returns a very good approximation of L^* in the form of a very informed upper bound for it. Furthermore, BEAUTY-PS utilizes only a tiny fraction of the expensive estimators, as it only estimates edges on the solution path. Thus, on average, BEAUTY with $l_{est} = 0, l_{prune} = \infty$ was able to generate a very accurate approximation of the optimal solution, though at the loss of guaranteed optimality, at minimal estimation effort overhead.

3.2.4 Different Accuracy Levels

Table 1 determines the accuracy range of estimators in our experiments. Indeed, for an edge e, the accuracy of its first cost estimate l_e^1 relative to its best estimate l_e^3 is $l_e^1 / l_e^3 = f_1 / f_3$. A high ratio of f_1 / f_3 implies that a cheap estimator yields a good approximation of the best estimate. It is thus interesting to test the sensitivity of the algorithms discussed in this section w.r.t. different accuracy levels.

To that end, we ran another experiment with the same setting as described before, in four domains (Barman, Settlers, Sokoban, Tetris), and with $f_1 \in \{10, 11, 12\}, f_2 \in \{f_1 + 1, f_1 + 2, f_1 + 3\}, f_3 \in \{f_2 + 1\}$, which resulted in significantly higher ratios of f_1 / f_3. Specifically, the range of f_1 / f_3 changed from $20\% - 60\%$ to roughly $71.43\% - 83.33\%$.

The results of expensive estimator usage (i.e., r_{L_3} of BEAUTY, A-BEAUTY-2 and

A-BEAUTY-10) are almost identical to the results reported in Table 2, with at most 1% difference in any entry. However, the convergence of A-BEAUTY-10 was faster, where the average number of iterations until convergence changed from 9 in the first experiment to 5.65 in the second experiment. This suggests that relatively accurate cheap estimators do not affect the number of expensive estimations required to achieve optimality, but reduce the number of iterations necessary for convergence.

Chapter 4

Optimally Solving Tightest Admissible Shortest Path

"It is the mark of an educated mind to rest satisfied with the degree of precision which the nature of the subject admits, and not to seek exactness when only an approximation is possible."

Aristotle

Based on:
Eyal Weiss, Ariel Felner, Gal A. Kaminka, "Tightest Admissible Shortest Path" In *Proceedings of the International Conference on Automated Planning and Scheduling: ICAPS 2024*, pp. 643-652, 2024.

4.1 Algorithms for TASP

As indicated in Corollary 1, we can obtain optimal solutions for TASP problems by optimally solving the corresponding SLB and SUB problems. SLB was already addressed here by introducing BEAUTY, a complete algorithm that finds optimal solutions for it. Hence, this section focuses on solving SUB, particularly

towards solving TASP in a manner that takes advantage of information already obtained during the solution of SLB.

In Subsection 4.1.1 we present BEAST (Branch&bound Estimation Applied to Searching for Top, Alg. 4), a complete algorithm that finds optimal solutions for SUB problems. BEAST extends UCS to dynamically apply cost estimators during a best-first search w.r.t. upper bounds of edge costs, and it aims to reduce the number of expensive estimators used, by utilizing both cheaper estimates and prior information on U^*. BEAST is thus particularly suitable to be used whenever prior information on U^* is available.

We note that BEAST has several analogies to BEAUTY, but also several fundamental differences which are due to the fact that *upper bounds tighten towards lower values* whereas *lower bounds tighten towards higher values*. Thus, cheaper (and looser) estimates cannot be used analogously to guide the search in minimization of tight upper bounds (SUB) compared to minimization of tight lower bounds (SLB). Specifically, BEAUTY first uses a looser lower bound to guarantee that a more expensive lower bound estimate is required. This can be done since if a path π_1 to a node n and its tightest path lower bound $l_{\Theta(\pi_1)}$ are known, and a new path π_2 to n is considered, then if a loose lower bound of $c(\pi_2)$ is already greater than $l_{\Theta(\pi_1)}$, then necessarily $l_{\Theta(\pi_2)} > l_{\Theta(\pi_1)}$ (i.e., π_1 is better). Hence, more expensive estimates for π_2 can be avoided. In contrary, this cannot be done analogously with looser upper bounds. Indeed, knowing that a loose upper bound of $c(\pi_2)$ is already greater than $u_{\Theta(\pi_1)}$ does not imply that $u_{\Theta(\pi_2)} > u_{\Theta(\pi_1)}$. Hence, more expensive estimates for π_2 cannot be avoided in the same way. For this reason, BEAST makes use of a combination of upper and lower bounds to try to avoid expensive estimates (see full description in Subsection 4.1.1).

In Subsection 4.1.2 we introduce BEAUTY&BEAST (Alg. 5), a complete algorithm that finds optimal solutions for TASP problems, that first uses BEAUTY to optimally solve SLB, and then uses BEAST, together with prior information already obtained on U^*, to optimally solve SUB.

Example 4 *Consider calling* BEAST *with* $u_{prune} = \infty$ *(i.e., base setting) on* P *from Example 1. Tracing its run, at the first iteration of the outer while loop* v_0 *is removed*

Algorithm 4 BEAST

Input: Problem $P = (G, v_s, V_g)$, where G is an estimated weighted digraph with cost estimators function Θ
Parameter: Threshold u_{prune}
Output: Path π, bound U^*

$g_u(s_0) \leftarrow 0$; OPEN $\leftarrow \emptyset$; CLOSED $\leftarrow \emptyset$;
Insert s_0 into OPEN with key $g_u(s_0)$
while OPEN $\neq \emptyset$ **do**
  $n \leftarrow$ Pop node n from OPEN with minimal $g_u(n)$
  if $Goal(n)$ **then**
    return $trace(n), g_u(n)$ // return π, U^*
  Insert n into CLOSED
  for each successor s of n **do**
    if s not in OPEN $\cup$ CLOSED **then**
      $g_u(s) \leftarrow \infty$
    $l(e) \leftarrow 0, u(e) \leftarrow 0$
    while $g_u(n) + l(e) < g_u(s)$ **and** $g_u(n) + l(e) \leq u_{prune}$ **and** estimators remain for $e = (n, s)$ **do**
      $l(e), u(e) \leftarrow$ Apply next estimator for e
    $\tilde{g}_u \leftarrow g_u(n) + u(e)$
    if $\tilde{g}_u < g_u(s)$ **and** $\tilde{g}_u \leq u_{prune}$ **then**
      $g_u(s) \leftarrow \tilde{g}_u$
      if s in OPEN **then**
        Remove s from OPEN
      Insert s into OPEN with key $g_u(s)$ and parent n
return $\emptyset, \infty$

from OPEN, $\theta^1_{e_{01}}, \theta^1_{e_{02}}$ and $\theta^2_{e_{02}}$ are invoked, and v_1, v_2 are inserted to OPEN with keys $4, 5$. *At the second iteration v_1 is removed from OPEN, $\theta^1_{e_{14}}, \theta^2_{e_{14}}$ are invoked, and v_4 is inserted to OPEN with key* 10. *At the third iteration v_2 is removed from OPEN, $\theta^1_{e_{23}}, \theta^2_{e_{23}}$ and $\theta^1_{e_{24}}$ are invoked, and v_3 is inserted to OPEN with key* 13. *At the forth iteration v_4 is removed from OPEN and* BEAST *returns* $\langle e_{01}, e_{14} \rangle, 10$.

4.1.1 The First Algorithm: BEAST

Algorithm 4 receives an SUB problem instance and one hyper-parameter u_{prune}. For simplicity we will first describe a *base case* where u_{prune} is set to ∞, and therefore has no effect and can be ignored. The relevant instructions using u_{prune}

are in Lines 12 and 15, and should be ignored for now. We will come back to this parameter later.

Base Setting. BEAST is structurally similar to UCS. It activates a best-first search process using the standard OPEN and CLOSED lists. Nodes n in OPEN are prioritized by $g_u(n)$ which is always equal to the optimal upper bound to node n along the best known path (similar to using $g(n)$ for ordering nodes in UCS in regular graphs, which is done according to optimal cost). The *best* such node n is chosen for expansion in Line 4, and its successors are added in the loop of Lines 8–19. When a goal node n is found in Line 5, the solution path π ending in n and its tight upper bound $g_u(n) = U^*$ are returned (Thm. 4).

The main difference of BEAST over UCS is in the *duplicate detection* mechanism performed when evaluating the cost of a new edge e that connects n to its successor s. In UCS, the exact edge cost $c(e)$ is immediately obtained and used to update the path cost that ends in s. In BEAST, we iterate over the different estimators θ_e^i for edge e (Lines 12–13). In each iteration $g_u(n) + l(e)$ serves as a lower bound for the tightest path upper bound of the path to s given the current estimator (Line 13). Namely, the tight upper bound is used up to node n and a lower bound is used for the edge e from n to s. Now such a path can be already pruned earlier if its current lower bound for the tight upper bound (using the current estimator) will not improve the best known path to s ($g_u(s)$). In that case we will not need to further activate more expensive estimators. Thus, if $g_u(n) + l(e) \geq g_u(s)$, the while statement (Line 12) ends. Then, ordinary duplicate detection is performed in Lines 14–19.

We emphasize that in case the while loop (Line 12) terminates due to $g_u(n) + l(e) \geq g_u(s)$ being satisfied, then necessarily $\tilde{g}_u = g_u(n) + u(e) \geq g_u(s)$ will be satisfied as well and the path will be (justifiably) pruned. On the other hand, notice that $u(e)$ cannot be used in place of $l(e)$ for early pruning, since upper bound estimates tighten towards lower values. See Example 4 for a demonstration of using BEAST in its base setting.

Remark 2 *In its base setting* BEAST *eventually uses the best estimates for edges leading to new nodes, thus it can be modified to directly jump to the best estimates in these*

cases. This was left out for simplicity of the pseudo-code.

Enhanced Setting. We now consider the enhanced setting where u_{prune} is set to some constant value (not ∞). In this setting u_{prune} serves as an upper threshold that limits the search, similarly to *Bounded Cost Search* Stern et al. (2014). This manifests in two aspects. First, u_{prune} is used as an upper threshold to exit the while loop (Line 12) and avoid activating more expensive estimators in case the tight upper bound to s is determined to be greater than u_{prune}. This is done in Line 12 where the condition $g_u(n) + l(e) \leq u_{prune}$ is tested, and if not fulfilled it breaks the loop. As explained in the base setting, $g_u(n) + l(e)$ serves as a lower bound to the tight upper bound to s and can thus facilitate early stopping. Second, u_{prune} is used as an upper threshold to prune (and not add to OPEN) any node with upper bound $> u_{prune}$. This is done in Line 15.

The primary purpose of using u_{prune} with $U^* \leq u_{prune} < \infty$ is to avoid applications of redundant (and potentially expensive-to-compute) estimators. Additionally, it can decrease the size of OPEN, which implies less insertion operations and cheaper insert/delete operations. Since U^* is unknown, setting this hyper-parameter to a meaningful value requires prior information. Practically, such information can be achieved by obtaining a suboptimal solution with $u_{sub} \geq U^*$, and using it to set $u_{prune} = u_{sub}$.

Finally, in case $U^* \leq u_{prune} < \infty$ is satisfied then a solution path π will be found (if a solution exists) and it will be returned together with its tight upper bound $g_u(n) = U^*$ (Thm. 4). On the other hand, in case $u_{prune} < U^*$ is satisfied then $\emptyset, \infty$ are returned (Thm. 5). See Example 5 for a demonstration of using BEAST in its enhanced setting.

Example 5 *Consider calling* BEAST *with* $u_{prune} = 4$ *(i.e., enhanced setting,* $u_{prune} < U^* = 10$*) on P from Example 1. Tracing its run, at the first iteration of the outer while loop* v_0 *is removed from OPEN,* $\theta^1_{e_{01}}, \theta^1_{e_{02}}$ *and* $\theta^2_{e_{02}}$ *are invoked, and* v_1 *is inserted to OPEN with key* 4*. At the second iteration* v_1 *is removed from OPEN,* $\theta^1_{e_{14}}$ *is invoked. At the third iteration OPEN is empty and* BEAST *returns* $\emptyset, \infty$*.*

Now consider calling BEAST *with* $u_{prune} = 11$ *(i.e., enhanced setting,* $u_{prune} \geq U^* = 10$*) on P from Example 1. Its run is identical to the run from Example. 4, with the same*

output, except that at the third iteration $\theta^2_{e_{23}}$ is not invoked.

Next, we provide the theoretical guarantees for BEAST.

Theorem 4 (Conditional Completeness and Optimality, Prob. 3) 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. First, it is straightforward to see that every node encountered by BEAST after the initial node is a successor of another node encountered during the search, as new nodes are only introduced in Line 8. Additionally, every node inserted into OPEN (except the initial node) is saved with a pointer to its parent node. Hence, whenever a goal node is found (at Line 5), it can necessarily be traced back to the initial node via a series of connected nodes, i.e., a valid solution path is returned at Line 6.

Second, BEAST inspects nodes that are removed from OPEN by best-first order w.r.t. upper bound of path cost. Since finding a goal node at Line 5 terminates the search, it is assured that the path leading to the first goal node found will be returned. Hence, the solution returned necessarily has the best (lowest) upper bound of path cost out of all the paths inspected by BEAST.

It remains to be shown that the search is systematic, namely that every path with tight upper bound up to u_{prune} is inspected by BEAST; and that every edge encountered during the search is either tightly (fully) estimated, or it is at least estimated in a manner that enables BEAST to determine that it is not part of the solution path.

Consider any successor s of a node n that is popped from OPEN. If the node s is encountered for the first time, then at first $g_u(s) \leftarrow \infty$ is set at Line 10, so the condition $g_u(n) + l(e) < g_u(s)$ at Line 12 is never satisfied. This means that the edge e leading to s will either be tightly (fully) estimated in case the current path to s has tight upper bound smaller or equal to u_{prune}, or otherwise the tight upper bound to s is greater than u_{prune}. Since $u_{prune} \geq U^*$ is assumed, this means that this path is not relevant and can be safely ignored. Indeed, in this case it is rightfully pruned in Line 15. If the node s was already encountered

earlier in the search, then the same mechanism described above applies, but additionally we have to consider the condition $g_u(n) + l(e) < g_u(s)$ at Line 12. In case it is not satisfied then this means that a better path was already found to s so it can be safely ignored, and indeed it is pruned in Line 15.

Lastly, since we have shown that the search up to tight upper bound u_{prune} is systematic, and considering that each edge has a finite number of estimators where each of them has finite runtime, BEAST necessarily terminates in finite time either when an optimal solution is found and returned (Line 6) with U^*, or when the search is exhausted up to tight upper bound u_{prune} and BEAST reports that no solution exists (Line 20). Note that in the latter case the assumption $u_{prune} \geq U^*$ implies that u_{prune} must have been set to ∞, so no solution at all exists. Hence, if $u_{prune} \geq U^*$ holds, BEAST is complete and returns an optimal solution. ■

Theorem 5 (Soundness, Prob. 3) *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.*

Proof. Following the proof of Thm. 4, since the search is systematic up to tight upper bound u_{prune}, it follows that if $u_{prune} < U^*$ holds then all paths with tight upper bound greater than u_{prune} will be pruned, and since every solution has at least tight upper bound U^*, the search will necessarily terminate with $\emptyset, \infty$. The other direction, i.e., $u_{prune} \geq U^*$, is assured by Thm. 4. ■

4.1.2 The Second Algorithm: BEAUTY&BEAST

Algorithm 5 receives a TASP problem instance, and returns an optimal solution for it (Thm. 6). It works by first calling BEAUTY on P (Line 1). If no solution is found then by the completeness of BEAUTY no solution at all exists, and $\emptyset, \infty$ are returned (Lines 2–3). Otherwise, the tightest upper bound of the solution found by BEAUTY, $u(\pi_{SLB})$, is obtained (Line 4). In case $L^* = u(\pi_{SLB})$ then necessarily $L^* = C^* = U^*$ and then π_{SLB} and $\mathcal{B}^* = 1$ are returned (Line 5–6). Otherwise, BEAST is called on P (Line 7) with $u_{prune} = u(\pi_{SLB})$, which is

Algorithm 5 BEAUTY&BEAST

Input: Problem $P = (G, v_s, V_g)$, where G is an estimated weighted digraph with cost estimators function Θ
Output: Path π^*, bound $\mathcal{B}^*$

$\pi_{SLB}, L^* \leftarrow$ Solve SLB for P using BEAUTY
if $\pi_{SLB} = \emptyset$ **then**
  return $\emptyset, \infty$
$u(\pi_{SLB}) \leftarrow u_{\Theta(\pi_{SLB})}$ // Get the upper bound of π_{SLB}
if $L^* = u(\pi_{SLB})$ **then**
  return $\pi_{SLB}, 1$
$\pi^*, U^* \leftarrow$ Solve SUB for P using BEAST, $u(\pi_{SLB})$
if $L^* = 0$ **and** $U^* > 0$ **then**
  return π^*, ∞
return $\pi^*, U^*/L^*$

necessarily greater or equal to U^*, and thus by Thm. 4 a shortest path tightest upper bound is returned. If $U^* > L^* = 0$ then the shortest path tightest upper bound that was found, π^*, is returned together with ∞ (Line 8–9). Otherwise π^* and $\mathcal{B}^* = U^*/L^*$ are returned.

Remark 3 *Note that in Alg. 5* BEAUTY *can be replaced by* any *algorithm that solves SLB optimally. Moreover, it can be replaced by an anytime algorithm, and then each time the lower bound or upper bound are improved (tightened) they can be translated to a tightened $\mathcal{B}$.*

Theorem 6 (Completeness and Optimality Prob. 1) BEAUTY&BEAST *is complete. Furthermore, if a solution exists for P then a tightest admissible shortest path π and $\mathcal{B}^*$ are returned (if $U^* > L^* = 0$ holds, then $\mathcal{B}^* = \infty$).*

The proof follows directly from the completeness and optimality of BEAUTY and BEAST, and from Thm. 2.

4.2 Empirical Evaluation for TASP

The theoretical guarantees of BEAST (base setting) and BEAUTY&BEAST (enhanced setting) assure optimality and completeness, but do not provide information about their runtime performance. We therefore empirically evaluate

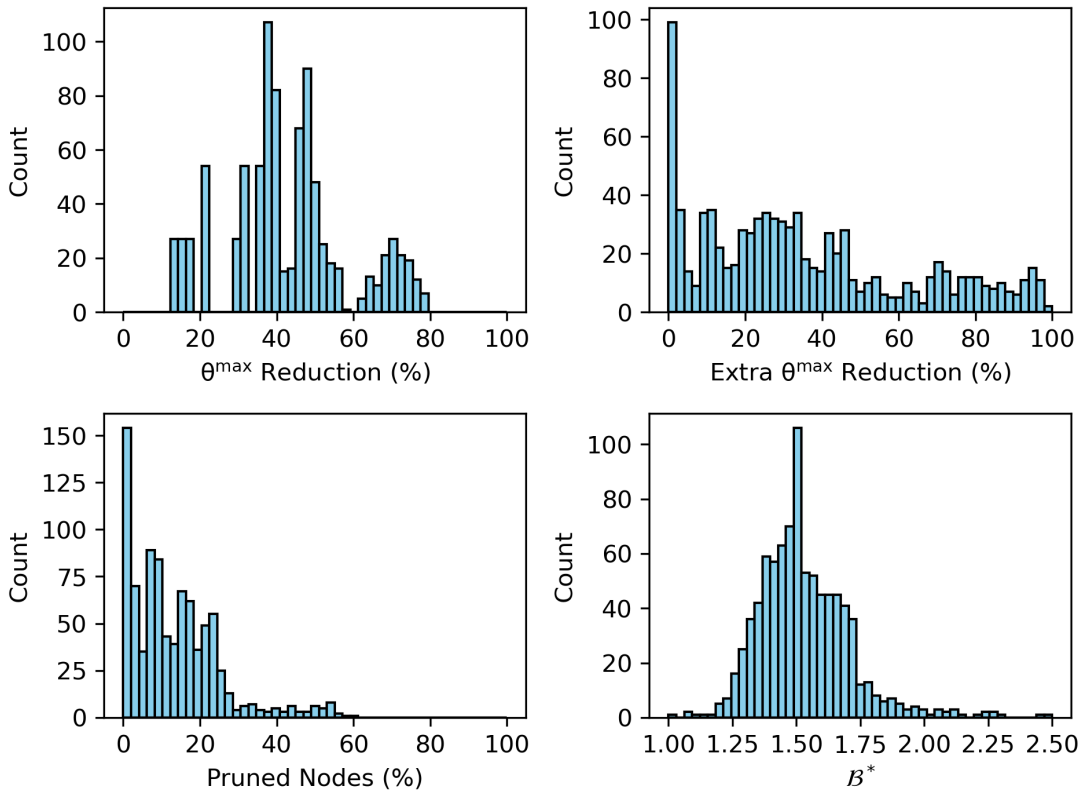


Figure 4.1: Histograms of $1 - (\theta^{max}(\text{BEAST } (\infty))/\theta^{max}(\text{EI-UCS}))$ [top left], $1 - (\theta^{max}(\text{BEAST } (u(\pi_{SLB})))/\theta^{max}(\text{BEAST } (\infty)))$ [top right], pruned nodes percentage for BEAST $(u(\pi_{SLB}))$ [bottom left], $\mathcal{B}^*$ [bottom right], based on all domains.

the algorithms in diverse settings, based on AI planning benchmark problems that were modified to have multiple action-cost estimators, so that these induce TASP problems.

The set of problems was taken from a collection of IPC benchmark instances[1]. Starting from the full collection, we selected all domains that use action costs, were part of an optimal track in IPC, and without duplication (e.g., only one version of the Elevators domain). Then, we created additional problems by using different configurations of costs. Lastly, for all domains and problems, we synthesized three estimators, using six parameters $f_1, ..., f_6$, according to the scheme described below.

[1]See `https://github.com/aibasel/downward-benchmarks`.

Domain	Instances	θ^{max} Reduction	Extra θ^{max} Reduction	Pruned Nodes	$\mathcal{B}^*$
Barman	135	39.46±3.22	7.00 ± 9.03	2.20 ± 2.80	1.49±0.12
Caldera	135	17.91±2.62	27.16± 5.20	8.28 ± 1.49	1.52±0.13
Elevators	135	70.79±4.14	69.24±24.73	30.34±15.07	1.56±0.26
Settlers	81	36.08±3.73	33.31± 8.77	16.26± 2.48	1.52±0.13
Sokoban	135	44.55±6.73	17.69±25.95	5.47 ± 8.31	1.54±0.21
Tetris	135	36.52±4.62	28.81±12.32	14.82± 7.21	1.54±0.20
Transport	135	50.55±3.36	62.40±19.79	17.95± 5.47	1.52±0.15
All domains (avg±std)	**891**	**42.64±15.88**	**35.08±27.69**	**13.40±11.75**	**1.53±0.18**
All domains (min–max)	**891**	**14.28–79.43**	**0.07–98.37**	**0.02–60.12**	**1.03–2.47**

Table 5: Summarized performance data of BEAST (∞), BEAST $(u(\pi_{SLB}))$, with breakdown by domains. θ^{max} is the number of expensive estimators. Column 3 refers to $1-(\theta^{max}(\text{BEAST } (\infty))/\theta^{max}(\text{EI-UCS}))$, Column 4 refers to $1-(\theta^{max}(\text{BEAST } (u(\pi_{SLB})))/\theta^{max}(\text{BEAST } (\infty)))$, Column 5 refers to pruned nodes out of generated nodes for BEAST $(u(\pi_{SLB}))$, Column 6 refers to $\mathcal{B}^*$. The entries for each domain (Rows 2–8) show average ± standard deviation. Cumulative results show average ± standard deviation (Row 9) and minimum–maximum (Row 10). All results are in percentages, except $\mathcal{B}^*$ values.

The original cost $c_{old}(e)$ (that is implied by the original domain and problem files) of each edge e was mapped to a new cost $c_{new}(e)$ that satisfies

$$c_{old}(e) \times f_3 \leq c_{new}(e) \leq c_{old}(e) \times f_4, \tag{4.1}$$

i.e., the exact value of $c_{new}(e)$ is contained in the interval $[c_{old}(e) \times f_3, c_{old}(e) \times f_4]$. Lower bounds were defined by

$$l_e^1 := c_{old} \times f_1, l_e^2 := c_{old} \times f_2, l_e^3 := c_{old} \times f_3, \tag{4.2}$$

with $f_3 \geq f_2 \geq f_1 \geq 1$, so that l_e^i is the i$^{\text{th}}$ lower bound (l_e^1 is the loosest and l_e^3 is the tightest). Upper bounds were analogously defined by

$$u_e^1 := c_{old} \times f_6, u_e^2 := c_{old} \times f_5, u_e^3 := c_{old} \times f_4, \tag{4.3}$$

with $f_6 \geq f_5 \geq f_4 \geq f_3$, so that u_e^i is the i$^{\text{th}}$ upper bound (u_e^1 is the loosest and u_e^3 is the tightest).

To diversify the estimator sets for different edges, the parameters for lower

bounds f_1, f_2, f_3 were taken from the sets

$$\begin{aligned} &f_1 \in \{1,2,3\}, f_2 \in \{f_1, f_1+1, f_1+2\}, \\ &f_3 \in \{f_2, f_2+1, f_2+2\}. \end{aligned} \tag{4.4}$$

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

$$\begin{aligned} &f_4 \in \{f_3+1, f_3+2, f_3+3\}, \\ &f_5 \in \{f_4, f_4+1, f_4+2\}, \\ &f_6 \in \{f_5, f_5+1, f_5+2\}. \end{aligned} \tag{4.5}$$

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

$$\text{Hash} = (c_{old}(e) + \text{seed}) \mod 27, \tag{4.6}$$

so that every value of Hash corresponded to one estimator configuration for the lower and upper bounds. Each problem was run once per seed, where all seeds from the set $[0, 26]$ were taken (namely, 27 seeds per problem). The full list of domains, problems and configurations that were used in the experiments is detailed in (link redacted for anonymity).

We note that the estimator configurations that were chosen according to the hash function of Eq. (4.6) guarantee that the same ground action, in different states, will have the same cost estimates. However, we also note that our algorithms can seamlessly handle state-dependent cost estimates.

BEAST and BEAUTY&BEAST were implemented as search algorithms in *PlanDEM*. All experiments were run on an Intel i7-1165G7 CPU (2.8GHz), with 32GB of RAM, in Linux. We also implemented *Estimation-time Indifferent* UCS (EI-UCS), a UCS algorithm that uses the most accurate estimate on each edge it encounters, to serve as a baseline for solving SUB.

We measured the performance of solving SUB via EI-UCS, BEAST (base setting)

and BEAUTY&BEAST (enhanced setting with information obtained from solving first SLB). We report the results (summarized in Table 5, extended in Fig. 4.1) from problem instances which all algorithms solved successfully, i.e., found optimal solutions, within 5 minutes. Overall, we report on a cumulative set of 891 problem instances, spanning 7 unique domains.

4.2.1 Base Setting vs. EI-UCS

In its base setting BEAST expands the same nodes as EI-UCS, but with potentially fewer expensive estimates (replaced by less expensive ones). Thus, we are interested in the relative savings that it achieves in practice. We denote the number of the *most expensive* estimators invoked during the search of an algorithm ALG by $\theta^{max}(ALG)$. Column 3 in Table 5 provides the relevant data: it shows $1 - (\theta^{max}(\text{BEAST } (\infty))/\theta^{max}(\text{EI-UCS}))$ in percentages, per domain (Rows 2–8), cumulatively by average±standard deviation (Row 9), and by range minimum–maximum (Row 10).

Roughly 40% of the expensive estimators are saved on average, with large differences across domains and problems. Over all, the range is from 14% to almost 80%. The top left plot in Fig. 4.1 shows the full empirical distribution (histogram), illustrating the high variability in the results. We suspect that different distributions of edge costs explain some of the variability, a topic left for future research.

4.2.2 Enhanced Setting vs. Base Setting

We can test the performance boost that can be achieved in practice from using BEAST in its enhanced setting with $u_{prune} = u(\pi_{SLB})$, by running BEAUTY&BEAST. Column 4 in Table 5 provides this data: the format is similar to that of Column 3, but instead it shows $1 - (\theta^{max}(\text{BEAST } (u(\pi_{SLB})))/\theta^{max}(\text{BEAST } (\infty)))$. It can be seen that roughly 35% of the expensive estimators are saved on average *w.r.t. to the base setting of* BEAST, again with very large differences across domains and problems, where over all problem instances the range is from 0% to around 98%. The top right plot in Fig. 4.1

shows the full empirical distribution, revealing that in the most common case the savings are rather low, but on the other hand in quite a few cases the savings achieved are very substantial.

Column 5 in Table 5 presents pruned nodes (out of generated nodes) for BEAST $(u(\pi_{SLB}))$, and it shows that roughly 15% of the generated nodes are pruned on average. Column 6 shows $\mathcal{B}^*$ (which is a property of the problem instance), where we can see that approximately it was on average 1.5 (the values in this column are not in percentages).

For both of these measures, we see high variability in the results: The range of pruned nodes (Column 5) is from 0% to 60%. The range of $\mathcal{B}^*$ over all problems was from 1 to 2.5. However, examining the distributions for both columns (Fig. 4.1; bottom left plot for pruned nodes, bottom right for $\mathcal{B}^*$), we see that the distributions are qualitatively very different. We will investigate this in future research.

We conclude that BEAST seems to offer significant empirical gains over EI-UCS, and that using the information obtained from solving SLB can, though not always, provide an additional significant performance boost.

Part II

Automated Planning with Multiple Cost Estimators

Chapter 5

Online Modeling for Offline Planning

"In preparing for battle I have always found that plans are useless, but planning is indispensable."

Dwight David Eisenhower

Based on:
Eyal Weiss, Gal A. Kaminka. "Position Paper: Online Modeling for Offline Planning" In *Workshop on Reliable Data-Driven Planning and Scheduling*. 2022.

5.1 Introduction

The definition and representation of planning problems is at the heart of AI planning research. Traditionally, planning (really, offline planning, i.e., planning before execution) has been defined as the process of generating a plan or policy that will take an agent that executes it to a state that achieves (or maximizes) its goals. Implicit in this definition is reliance on an *action model*, created offline (ahead of the planning process). This allows the process to consider the actions the agent may take at any given state, what their effects will be, and at what costs.

The representation of action models given to the planner is a main topic of research. Over the years, steady efforts have been made to develop action model representations that are both general and expressive using declarative languages Fox and Long (2003); Baier et al. (2008); Eyerich (2013). Optimized open-source software implementations (e.g., Helmert (2006)), which admit these representations, can use sophisticated search algorithms Lipovetzky and Geffner (2017), and efficient domain-independent heuristics (see Ghallab et al. (2016)), to solve challenging planning problems in a variety of domains.

However, despite the maturity of the field, AI planning technology is still rarely used outside the research community. After decades of advances improving action model representations, they remain disconnected from many real world requirements Blythe (1999); McCluskey (2003); Boddy (2003); Rintanen (2015). In some cases, the effects of actions are too complex to describe declaretively, or there may be uncertainty as to the effects or their cost. In some cases, models may only be available in black-box form (e.g., when learned Arora et al. (2018)); this limits their usability in different planners. Furthermore, some models are computationally expensive to build in advance of the planning, so preparing informative action models may be unfeasible. For instance, declaring the cost of possible ground actions using external sources, ahead of the planning process, is practically impossible in real-world domains.

Semantic attachments, which were first introduced in Weyhrauch (1980) and incorporated in PDDL by Dornhege et al. (2009), is an important concept that offers a partial solution to some of the problems raised above. It allows combining external procedures that supplement declarative models by performing condition checks and calculating action effects, thus offering access to modeling of complex mathematical functions and postponing calculations involving them to the planning process. However, this ignores the run time and of external procedures, and the uncertainty they may represent. When using data-driven models for planning, one has to address these issues, so as to decrease the computational burden of modeling (while planning), and increase the reliability of the plan found.

We argue that the common cause for the above difficulties is that the modeling

process is assumed to have taken place and completed prior to the planning process, i.e., *offline modeling* for *offline planning*. The current approach to planning requires that the modeling process, whether carried out manually or by learning, generates a single set of action models, which the planner later uses during planning. This raises the following challenges inherent to this approach:

Waiting for the perfect PDDL. To be general, planners must be able to work with action model description languages that capture complex effects and conditions. Yet despite decades of advances in PDDL, that there are still many cases in which its declarative expressiveness is not sufficient, e.g., because of limited mathematical operator set, or because some state factors are not easily described by truth-values or even purely numerically. Of course, we should continually improve and extend PDDL, but this is a slow process, and even slower in terms of building planners able to support advanced features. Even today, few if any planners admit the full set of features of the latest PDDL.

Inability to work with multiple languages. As the perfect modeling language is still in our future, existing planners in challenging areas of significant practical value, use hybrid models. For example, in Task and Motion Planning (TAMP), both a task description must be provided (e.g., in PDDL), as well as description of geometric and kinematic considerations of motion Garrett et al. (2021).

Early commitment to modeling choices. Even supposing a single language is sufficient, one may create alternative models, at different levels of detail or accuracy, for the same action. When the modeling choice is made ahead of the planning process, a commitment must be made to the modeling level ahead of the planning process. Conservatively, one must choose the most detailed and accurate model, but this often entails significant planning computations later on. Thus, for instance, the use of multiple (increasingly accurate) estimators for cost or uncertainty is impossible.

Early commitment to modeling computation. The last point also ties to this one, which is the commitment ahead of planning to a single model per action, regardless of the run time it consumes. In a fully declarative setting, this means conducting all modeling computations in advance, which might be impracticable. For example, if the model needs the time it takes to drive a specific road, then this information must be determined ahead of the planning (e.g., by accessing Google Maps). Even a 50ms query is a long process by computational standards, and certainly when every such possible ground action must be evaluated ahead of planning. Even when semantic attachments are used, early commitment to the model might incur significant overhead in case lighter models can be used for some actions (possibly offering reduced, but sufficient, accuracy).

Difficulty in using non-declarative, learned, models. Models learned from data almost always include some form of uncertainty. Moreover, the accuracy of learned models often depend on the run time allocated for them (e.g., an anytime procedure that continually improves until it is stopped). When the models are fixed in advance of planning, the planner has limited ability to affect the overall uncertainty of the plan it returns, thereby reducing its effectiveness to robustly cope with learned models.

We therefore suggest to change the AI planning process, such that is carries out *online modeling* in offline planning, i.e., the use of action models that are computed or even generated as part of the planning process, as they are dynamically accessed. This generalizes the existing approach (offline modeling). Presently, there are two major lines of research taking this approach: *planning with simulators* Francès et al. (2017), and domain-specific attempts, often within TAMP. While these are good examples and successful in some applications, they do not offer a full solution to the gap in problem modeling. Indeed, *planning with simulators* sacrifices much mathematical structure—inherent in declarative action models—rendering many known heuristics inapplicable. TAMP is domain-specific, and thus does not offer a high level of generality. More generally, there are theoretical questions that arise from this changed perspective on planning, which have not been addressed.

We propose to re-formulate the definition of planning, to allow online modeling. Rather than relying on a set of pre-computed action models, the planner should accept a set of action model estimators which are called to dynamically generate action models as needed. Multiple models can even be generated for the same action, thus allowing reasoning about models at different levels of resolution (and possibly computational cost).

The formulation admits existing efforts as special cases. Offline modeling is a special case where there is a single estimator for every action, and it uses a declarative description. Planning with semantic attachments is a special case where *modeling computation* is conducted online, but modeling choices are prefixed. Planning with simulators is a special case where *the effects* of actions are generated by calling simulators. For hybrid TAMP planners, the applicability of a motion (preconditions), or its duration (cost) are computed by calling a local motion planner.

The proposed definition admits novel planning processes. For example, under this formulation the initial declarative model may be viewed as a first-order approximation of the correct model, where each application of an estimator improves the approximation. This closely follows Rao's vision of model-lite planning Kambhampati (2007) for working with approximate models in a systemic manner. It also resembles the work on planning and acting Ghallab et al. (2016) which aims for continual refinement of deliberative models during the execution of the plans (i.e., online planning); in contrast, we focus on offline planning.

5.2 Online Action Models

The commonly-accepted definition for planning problems is a tuple $P = (\mathcal{I}, \mathcal{G}, \mathcal{A})$, whose components represent the initial state(s), the goal states, and the set of action models, respectively. Each action model $a \in \mathcal{A}$ has associated preconditions, $PRE(a)$, effects $EFF(a)$, cost $c(a)$, and sometimes duration or effect probabilities. In other words, every action model in a is a complex object, which can be queried for information about the action. Commonly, the answers

to the queries remain static throughout the planning process.

We propose to generalize the approach to offline planning, so that it allows the answers to the queries to dynamically change during the planning process. The proposed modification in definition does not, at a high level, change the tuple components. Instead, the action model objects $a \in \mathcal{A}$ are considered to be dynamic estimators, rather than static. But this profoundly changes our conception of planning.

We point out several appealing properties of the generalized approach. First, any kind of estimator can be used, so there are no restrictions on the type of data being processed during planning, nor on the mathematical operators being utilized, and in particular estimators can be black-box. Second, action model estimators can be based on the outcome of domain-specific planning systems (such as motion planners), so that a form of hierarchical planning may take place, *allowing the flow of bottom up information during planning*. Third, mixed declarative-procedural problem formulations are supported, allowing declarative-based state-of-the-art domain-independent heuristics to retain relevance. Lastly, model uncertainty can be systematically controlled to meet target plan accuracy, while offering significant potential savings on redundant modeling time.

We distinguish families of action model estimators. Two of specific interest are those that deal with estimation of the symbolic structure of the model, i.e., the preconditions and effects (e.g., as in planning with simulations), and those that estimate the numeric parameters of the actions, such as costs, duration, or probabilities of effects. Some TAMP planners take advantage of both types. Other planners, e.g., those that interleave calls to external sources of information as part of the planning, are equally described in this definition.

We focus here on one specific—novel—type of planning problems which can be described as online modeling for offline planning. In these, the planner begins with an initial action model, and then utilizes external sources of information for model completion, ad hoc. Concretely, we focus on the following case:

- The problem's symbolic *structure* is described using declarative action

models with structural preconditions and effects (e.g. via predicates). These remain static.

- The *numeric* model parameters are estimated online, with increasing accuracy, by letting the planner call estimators that provide information about their values.
- The planner is given an acceptable accuracy for the sought-after plan, allowing the planner to trade-off accuracy vs. computation time.

To make this explicit, we add a component to the the planning problem definition. In addition to the initial state, goal and the set of action models, we add Θ, a set of action model parameter *estimators*, so that calling such an estimator with an action model returns an updated model. The planning problem explicitly denotes this as a tuple $P = (\mathcal{I}, \mathcal{G}, \mathcal{A}, \Theta)$.

The immediate implication of the suggested approach is an enhanced ability to represent and solve planning problems, as clearly every declarative representation (that so far was constructed prior to planning) can be completed incrementally by the planner, given appropriate external modules. The price paid is increased planning time, due to additional computational effort spent on refining the model. This trade-off is typical for problem generalization, as it entails solving a harder problem. The main challenge that arises is thus to develop computationally efficient planners, able to balance resource allocation between search effort and model refinement effort. We discuss this in the next section.

5.3 When are Dynamic Models Preferable?

We focus on automated acquisition of action model parameters, where the structural elements defining the problem are constructed offline. Our target scenario is characterized by problems with structure that may be well described by a factored representation (namely using action templates), but where every ground action potentially has a unique set of parameters (e.g., cost, duration, probabilities etc.) that is not easily defined using a factored representation. Hence, obtaining numeric values for each action is done separately, so that ev-

ery action requires a function call—an invocation of an action model estimator. We use the term estimator as it generalizes a procedure which simply retrieves values to also account for potential parametric uncertainty. Additionally, we permit several distinct estimators per action, that provide a range of accuracy levels and running times.

For a given planning problem, we denote by

$$\mathcal{A} = \{a_1, \ldots, a_n\}, \Theta = \{\theta_1^1, \ldots, \theta_1^{k_1}, \ldots, \theta_n^1, \ldots, \theta_n^{k_n}\} \tag{5.1}$$

the set of ground actions and the set of action model estimators, respectively, where θ_i^j stands for the jth estimator of action a_i, and by $t(\theta_i^j)$ the run time of θ_i^j. W.l.o.g. we assume that $t(\theta_i^q) \leq t(\theta_i^m)$ for $q < m$. Using these notations we can express the run time required for constructing a full offline model:

$$T_{offline}^{modeling} = \sum_{i=1}^{n} t(\theta_i^{k_i}). \tag{5.2}$$

Note that if modeling has to be completed before planning then it has to be conservative, i.e., the best estimates available for every action must be used. Otherwise, the potential quality of plans, that can be found by a planner using these models, will decrease.

Examining $T_{offline}^{modeling}$, one can recognize that it may in fact be unknown a priori. While n is always known, the running times of the estimators might not be known in advance. However, it is reasonable to assume that some knowledge about them is available (such as their order of magnitude), so at least approximating $T_{offline}^{modeling}$ is plausible. Hence, when $t_{avg}^{best} := \frac{1}{n} T_{offline}^{modeling}$ is projected to be high, and n is not small, $T_{offline}^{modeling}$ might be very demanding. To give a concrete example, consider the case that an estimate is obtained via internet access, so network delay dominates estimator run time. In such a case $t_{avg}^{best} \approx 100$ms is a typical value (see e.g., https://wondernetwork.com/pings). As for n, benchmark problems contain up to 10^6 ground actions (and even more in extreme cases) Gnad et al. (2019). Thus, $T_{offline}^{modeling}$ might take dozens of hours. It is clear that in such cases there is a strong incentive to try to reduce modeling time.

We now turn to express the run time required for modeling in the case of dynamic estimation:

$$T^{modeling}_{dynamic} = \sum_{i:a_i \in \mathcal{A}_{actual}} t(\theta^{actual}_i), \tag{5.3}$$

where $\mathcal{A}_{actual}$ is the set of actions that require estimates during planning, and θ^{actual}_i are the estimators that are applied during planning. We highlight that these terms are planner-dependent, and they are also clearly unknown prior to planning. Nevertheless, it is well known that many planning problems could be solved while exploring only a small fraction of the problem state space, and correspondingly considering only a small part of all possible actions.

Therefore, even without knowing the exact value of $|\mathcal{A}_{actual}|$, it is reasonable to expect for a significant reduction in modeling time. In addition, it might not be necessary to use the most time-consuming estimator for each action encountered during planning, which implies another potential reduction in modeling time if $t^{actual}_{avg} := \frac{1}{|\mathcal{A}_{actual}|} T^{modeling}_{dynamic} \ll t^{best}_{avg}$. Overall we may conclude that $T^{modeling}_{dynamic} \leq T^{modeling}_{offline}$, and in many cases it might be that $T^{modeling}_{dynamic} \ll T^{modeling}_{offline}$.

So far we have seen that when considering modeling time alone it is always favorable to use dynamic estimation. However, ad hoc modeling may incur additional planning time, as some algorithmic mechanism would have to decide when to apply estimation, effectively adding a computational overhead. We denote

$$\Delta_{modeling} := T^{modeling}_{dynamic} - T^{modeling}_{offline}, \Delta_{planning} := T^{planning}_{dynamic} - T^{planning}_{offline} \tag{5.4}$$

as the differences in running times of modeling and planning (not including run time of estimators) correspondingly due to using dynamic model construction. Considering the combined run time of modeling and planning, we can determine that dynamic is computationally preferable to offline in case $|\Delta_{modeling}| > |\Delta_{planning}|$.

This criterion formally requires empirical testing per problem and planner, but may be practically useful when $T^{modeling}_{dynamic} \ll T^{modeling}_{offline}$ is projected, as it is reasonable to assume that $T^{planning}_{dynamic}$ and $T^{planning}_{offline}$ are on the same order of magnitude.

In conclusion, when $|\mathcal{A}_{actual}| \ll n$ or $t_{avg}^{actual} \ll t_{avg}^{best}$ (or both) are projected, it is advantageous to turn to online modeling.

Chapter 6

Automated Planning with Multiple Cost Estimators

"A wealth of information creates a poverty of attention."

Herbert Alexander Simon

Based on:
Eyal Weiss, Gal A. Kaminka. "Planning with Multiple Action-Cost Estimates" In *Proceedings of the International Conference on Automated Planning and Scheduling: ICAPS 2023*, pp. 427-437. 2023.

6.1 Introduction

AI planning applications require computing the costs of ground actions to optimize plans. The costs are computed by the planner for concrete ground actions that it considers adding to the plan, as part of the planning process. Typically, they are computed from a domain action model provided as input to the planner (e.g., using PDDL McDermott et al. (1998); Fox and Long (2003)). The runtime of action cost computation is generally considered to be negligible.

However, the computation of action costs can incur significant computational expense, to the point it becomes a major component in the overall planning

runtime. This happens when the computation is inherently expensive, e.g., as in the case of robot task and motion planning, where the planning process considers geometric and kinematic constraints on movement in continuous environments, and repeatedly computes distances and potential collisions LaValle (2006); Garrett et al. (2021); Dellin and Srinivasa (2016); Narayanan and Likhachev (2017); Mandalika et al. (2019). Cost computation runtime can also increase by the use of external black-box action models (including action costs) that are called by the planner as needed Dornhege et al. (2009); Gregory et al. (2012a); Francès et al. (2017); Allen et al. (2021).

Transportation logistics planning is an example domain where the two sources of expensive computation of action costs combine in real-world applications. The domain deals with planning efficient routes for trucks and payloads. A simple form of it is captured in the *Transport Sequential* IPC domain Coles et al. (2012), where costs are assumed to be given in the PDDL problem description, and their computation is immediate. But in actual commercial applications (e.g., TruckNet (2021)), *the route cost computation is expensive in itself*, as the system takes into account also truck maintenance and insurance costs, driver salaries, fuel prices in different locations, legal speed limits, time-of-day, and geometrical constraints such as elevation and road curvature. The travel time between cities can also be computed by queries to an online service such as Google Maps, an *external action model*. The online query time is measured in milliseconds per query; far from negligible.

To reduce the computational expense in computing action costs, previous work explored mainly two approaches. The first focuses on minimizing the number of independent expensive cost computations that must be carried out Dellin and Srinivasa (2016); Narayanan and Likhachev (2017); Mandalika et al. (2019). The second, exemplified by the D^* Stentz (1994)) and *Lifelong Planning* A^* Koenig et al. (2004) algorithm families, minimizes cost computations when previously-computed costs change.

We propose an alternative approach building on the key insight that the computation of action costs can be spread across *multiple* procedure calls. A first rough estimate can be generated quickly, while future calls to cost computation

procedures can increasingly improve the estimation, potentially taking longer to compute. This approach follows the line of the recently suggested concept of dynamic estimation during planning Weiss (2022); Weiss and Kaminka (2022).

In the case of the transportation logistics example, the travel time between two cities can be estimated quickly—but inaccurately—from their fixed geographical distance or mean travel times computed once, offline (e.g., the way a *transport sequential* problem file includes fixed costs). Online queries and expensive computations of travel time based on elevation and curvature can serve as more accurate—and more costly—estimates. More parameters can be taken into account in increasingly more expensive estimator procedures, resulting in more accurate estimates.

We therefore define a novel planning problem, *planning with multiple action cost estimates* (P-MACE). This is a generalization of classical planning problems, where every ground action can have multiple cost estimators. Each estimator provides a lower and upper bound on the actual cost, and the estimators are ordered by increasing computational expense. A P-MACE planner's task is to efficiently find a plan *meeting a target bound on the optimal cost*. This definition generalizes previous work in motion planning Dellin and Srinivasa (2016); see details in a separate section.

We present ACE (A^* with Cost Estimation), a generalization of A^* Hart et al. (1968) that allows a planner to efficiently select estimators so to generate plans meeting the target bound. ACE is sound (finds a correct plan) and complete (will always find a plan if one exists). Its optimality is not dependent only on the heuristic used, but also on the available estimators. In case every action cost can ultimately be estimated exactly, ACE is optimal. In other cases, it may fail to find plans meeting the target bound. To augment ACE, we present a post-search procedure that improves the accuracy of non-optimal solutions that do not meet the target bound. We implemented ACE in the Fast Downward planner Helmert (2006). Extensive experiments with 6600 planning problems generated from IPC benchmarks shows dramatic reduction in runtime compared to available alternatives, while maintaining estimated costs well within target bounds. The savings are also clear in an experiment with real-world data for the Ontario

province (Canada) using Google Maps query times.

6.2 Classical Planning

We use a simple notation for deterministic planning based on finite-state automaton Ghallab et al. (2016). A *planning domain* is a 4-tuple $\Sigma = (S, A, \gamma, c)$, where S is a finite set of system states, A is a finite set of (ground) actions, $\gamma : S \times A \rightarrow S$ is a partial function called the state-transition function, and $c : S \times A \rightarrow \mathbb{R}^+$ is a partial function called the cost function, that has the same domain as γ. If $\gamma(s, a) = s'$ is defined, then a is said to be applicable in s, and its cost $c(s, a) < \infty$ is called the transition cost. For simplicity we take $c(s, a)$ to be $c(a)$, i.e., we consider the costs of ground actions rather than transition costs. This is common practice and serves here for easier presentation. Nevertheless, the techniques presented below fully support dealing with transition costs.

Definition 5 *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 an initial state, and $S_g \subset S$ is a set of goal states. A* **solution** *for $\mathcal{P}$ is a plan $\pi = \langle a_1, ..., a_n \rangle$, a finite sequence of actions, s.t. $a_1 \in \pi$ is applicable in s_0; for all $i \leq n$, a_i is applicable in s_{i-1} and $s_i = \gamma(s_{i-1}, a_i)$; and $s_n \in S_g$. The plan length is $|\pi| = n$, and its cost is $c(\pi) = \sum_{i=1}^{n} c(a_i)$. A* **cost-optimal solution** *is a plan π^* that satisfies*

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

The directed graph $\mathcal{G}_\Sigma = (\mathcal{V}, \mathcal{E}, w)$ induced by the planning domain Σ is a *weighted digraph*, where: $\mathcal{V}$ is a set of vertices, each corresponding to a state $s \in S$, and $\mathcal{E}$ is a set of edges, s.t. $e = (s, \gamma(s, a)) \in \mathcal{E}$ iff an action $a \in A$ is applicable in s. For each edge $e \in \mathcal{E}$ the weight (cost) is given by $w(e) = c(a)$. For simplicity, we use $c(e)$ for edge weights.

We note that in the general case, it may be that two distinct actions a_1, a_2 applied in a state s both result in the same state, $\gamma(s, a_1) = \gamma(s, a_2)$, i.e., multiple edges connect the same source and target vertices, which would make $\mathcal{G}_\Sigma$ a multigraph. However, for simplicity of the presentation, we consider the planning

domains in the chapter to have the property that every two actions applied in the same state result in two distinct states. Therefore, there is a one-to-one correspondence between a planning domain Σ and the graph $\mathcal{G}_\Sigma$.

6.3 Planning with Action Cost Estimators

This section introduces mathematical definitions and notations for discussing action cost estimators. We then pose a novel planning problem that considers multiple cost estimators, generalizing over problems with exact costs, in the same way that EWDG extend weighted directed graphs. We conclude with a theoretical analysis that lays the basis for developing algorithms that solve the newly suggested problem.

6.3.1 Multiple Action-Cost Estimates

Mathematical Setting.

We begin by associating a set of cost estimators with every ground action in a planning domain Σ (each edge e in $\mathcal{G}_\Sigma$):

Definition 6 *The* cost estimators function Θ *for a planning domain* Σ *maps each edge* $e \in \mathcal{E}$ *to a finite and non-empty ordered set of estimators,*

$$\Theta(e) = (\theta_e^1, \ldots, \theta_e^{k(e)}), k(e) \in \mathbb{N},$$
$$s.t., \forall e, i, \theta_e^i : \emptyset \rightarrow \mathbb{R}^+ \times \mathbb{R}^+,$$

where each estimator θ_e^i *defines an accuracy interval, and* $\Theta(e)$ *is ordered by increasing estimation times (which are guaranteed to be finite).*

Throughout this chapter we make the following assumptions about cost estimators functions:

1. Each estimator provides finite bounds for the true cost, i.e., $\theta_e^i = (c_{min}^i(e), c_{max}^i(e))$ with $0 \leq c_{min}^i(e) \leq c(e) \leq c_{max}^i(e) < \infty$.
2. Positive costs have positive bounds, namely $c(e) > 0$ implies $c_{min}^i(e) > 0$.

3. Estimation bounds tighten with increasing order of estimators, so that the intervals induced by θ_e^j, θ_e^i satisfy $[c_{min}^j(e), c_{max}^j(e)] \subseteq [c_{min}^i(e), c_{max}^i(e)], \forall e, i < j$.

We note that zero costs are allowed, and that Assumptions 1 and 2 together imply that zero costs are identified by estimators, i.e., $c(e) = 0$ iff $c_{min}^i(e) = 0$. We also note that Assumption 1 can be relaxed to allow infinite values, and Assumption 2 can be removed entirely, yet they both simplify the exposition.

The Planning Problem.

The definition of planning changes when Θ is considered. The notion of a solution plan remains (sequence of actions from s_0 to a goal state). However, the optimal plan cost, that depends on exact costs, may not be computable. This is because Def. 6 and Assumptions 1–3 allow the exact cost of any edge to remain unknown, even after utilizing all possible estimates for that cost. Hence, we define optimality w.r.t. a bound $\mathcal{B}$ on the optimal (possibly unknown) cost, so estimation uncertainty is taken into account. This is formalized as follows.

Definition 7 *A* 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. A $\mathcal{B}$-**optimal solution** to $\mathcal{P}$ is a plan $\pi^{\mathcal{B}}$ that satisfies*

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

with c^* being the optimal cost and $\mathcal{B} \geq 1$.

P-MACE generalizes classical planning (Thm. 7) in two aspects: first, it allows the cost of an action to be unknown, yet quantified by a bounding interval; and second, it permits multiple time-consuming cost estimators per action. The implications of latter generalization is that planners can exploit this flexibility when searching for a solution by spreading estimation time across cheaper alternatives, when lower accuracy is deemed sufficient, in order to save runtime.

Theorem 7 (Generality) P-MACE *problems are a generalization of classical planning problems.*

Proof. *Any* classical planning problem can be formulated as a P-MACE problem, by considering the special case where each edge has one estimator (i.e., $k(e) = 1$ for every e), that returns the true cost (namely, $c^1_{min}(e) = c(e) = c^1_{max}(e)$), with no relaxation for plan optimality (which means that $\mathcal{B} = 1$). ■

6.3.2 Plan Cost Uncertainty Quantification

Naively, $\mathcal{B}$-optimal planning can be carried out by an *estimation-indifferent* process, that ignores computational expense and evaluates all estimators for any action a it considers for a solution. However, unless estimation time is negligible, the planner should instead carefully select the estimators to be applied.

Let Θ_Σ be the set of all estimators, for all edges in $\mathcal{G}_\Sigma$. A subset Φ of Θ_Σ is *valid* if it contains at least one estimator per edge. Given a plan $\pi = \langle a_1, ..., a_n \rangle$ and a valid $\Phi \subseteq \Theta_\Sigma$, the *plan lower bound* and *plan upper bound* w.r.t. Φ are defined (respectively) as

$$c^\Phi_{min}(\pi) := \sum_{i=1}^{n} c^i_{min,\Phi},\ c^\Phi_{max}(\pi) := \sum_{i=1}^{n} c^i_{max,\Phi}, \tag{6.1}$$

where $c^i_{min,\Phi}$ and $c^i_{max,\Phi}$ denote the *tightest* lower and upper bound estimates of a_i obtained from Φ. The cost uncertainty ratio of π w.r.t. Φ is then

$$\eta(\pi) := \frac{\sum_{i=1}^{n} c^i_{max,\Phi}}{\sum_{i=1}^{n} c^i_{min,\Phi}} = \frac{c^\Phi_{max}(\pi)}{c^\Phi_{min}(\pi)} \tag{6.2}$$

We define $\eta(\pi)$ to be 1 in case $\sum_{i=1}^{n} c^i_{min} = 0$, as this implies $c(\pi) = 0$, meaning that there is no uncertainty.

The choice of Φ directly impacts the cost uncertainty of a plan π, since

$$c^\Phi_{min}(\pi) \leq c(\pi) \leq c^\Phi_{max}(\pi) = c^\Phi_{min}(\pi) \times \eta(\pi) \tag{6.3}$$

The *optimal* plan cost lower bound $c^{*\Phi}_{min}$, and the *optimal* plan cost upper bound $c^{*\Phi}_{max}$ are the minimal plan cost lower and upper bounds over all plans for $\mathcal{P}$. An *optimal optimistic plan* π_{opt} and an *optimal pessimistic plan* π_{pes} are plans that

satisfy $c^{\Phi}_{min}(\pi_{opt}) = c^{*\Phi}_{min}$, $c^{\Phi}_{max}(\pi_{pes}) = c^{*\Phi}_{max}$.

Thm. 8 shows that the cost of an *optimal* optimistic plan is bounded by its cost uncertainty ratio and the optimal (unknown) cost c^*. Corollary 2 then shows this leads to being able to efficiently find $\mathcal{B}$-optimal plans, by searching for effective Φ.

Theorem 8 (Bound) *Given a* P-MACE *problem $\mathcal{P}$, an optimal optimistic plan π_{opt} w.r.t. any valid $\Phi \subseteq \Theta_\Sigma$, satisfies the following relation for its cost:*

$$c(\pi_{opt}) \leq c^* \times \eta(\pi_{opt}). \tag{6.4}$$

Proof. Denote by (c^i_{min}, c^i_{max}) the lower and upper bounds, for the cost of the ith action in π_{opt}. By definition of π_{opt} as an optimal optimistic plan w.r.t. Φ, it follows that

$$\sum_{i=1}^{n} c^i_{min} \leq c^*. \tag{6.5}$$

From the right inequality of (6.3), we get

$$c(\pi_{opt}) \leq \sum_{i=1}^{n} c^i_{max} = \sum_{i=1}^{n} c^i_{min} \times \eta(\pi_{opt}). \tag{6.6}$$

Using Inequality (6.5) to obtain an upper bound of the right hand side of Inequality (6.6) yields Inequality (6.4). ■

Corollary 2 *It follows from Thm. 8 that an optimal optimistic plan π_{opt} w.r.t. any valid $\Phi \subseteq \Theta_\Sigma$ with $\eta(\pi_{opt}) \leq \mathcal{B}$ is $\mathcal{B}$-optimal. This implies that we can reduce planning time by efficiently finding Φ that achieves this.*

6.4 ACE: A^* with Cost Estimation

We present ACE, a generalization of A^*, that solves P-MACE problems while reducing the number of unnecessary cost estimations. Based on Corollary 2, ACE uses the target suboptimality factor $\mathcal{B}$ to restrict the number of estimates used during the search for plans, comparing $\mathcal{B}$ to $\eta(p)$ for every path p being

considered.

ACE (Alg. 6) is given a P-MACE problem $\mathcal{P}$, a target $\mathcal{B}$, a heuristic h and a procedure GetEstimator, which incrementally selects estimators $\theta_e^i \in \Theta(e)$ for an edge e (see details below). ACE then tries to find a $\mathcal{B}$-optimal plan, and outputs the plan π it finds and reports its achieved cost uncertainty ratio $\eta(\pi)$. If no plan is found, $(\emptyset, \infty)$ is returned.

Algorithm 6 ACE

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

$g_{min}(s_0) \leftarrow 0$; $g_{max}(s_0) \leftarrow 0$
OPEN $\leftarrow \emptyset$; CLOSED $\leftarrow \emptyset$
Insert s_0 into OPEN with $f(s_0) = h(s_0)$
while OPEN $\neq \emptyset$ **do**
  $n \leftarrow$ best node from OPEN
  if $Goal(n)$ **then**
    return $trace(n), g_{max}(n)/g_{min}(n)$
  Insert n into CLOSED
  for each successor s of n **do**
    if s not in OPEN $\cup$ CLOSED **then**
      $g_{min}(s) \leftarrow \infty$
    $\eta \leftarrow \infty$; $\underline{g} \leftarrow g_{min}(n)$
    $\theta \leftarrow \mathsf{GetEstimator}(e = (n, s))$
    while $\eta > \mathcal{B}$ **and** $\underline{g} < g_{min}(s)$ **and** $\theta \neq \emptyset$ **do**
      $\underline{c}, \bar{c} \leftarrow \text{apply}(\theta)$
      $\underline{g} \leftarrow g_{min}(n) + \underline{c}$; $\bar{g} \leftarrow g_{max}(n) + \bar{c}$
      $\eta \leftarrow \bar{g}/\underline{g}$
      $\theta \leftarrow \mathsf{GetEstimator}(e)$
    if $\underline{g} < g_{min}(s)$ **then**
      $g_{min}(s) \leftarrow \underline{g}$; $g_{max}(s) \leftarrow \bar{g}$
      if s in OPEN $\cup$ CLOSED **then**
        Remove s from OPEN and CLOSED
      Insert s into OPEN with $f(s) = g_{min}(s) + h(s)$
return $\emptyset, \infty$

We explain how ACE works by comparing it to A^*. ACE uses accumulated bounds g_{min} and g_{max}, instead of the accumulated cost g. Similarly, f-values

are computed by $f(s) = g_{min}(s) + h(s)$, (i.e., g_{min} replaces g). There are two important deviations from A^*: in computing the heuristic values, and in estimating the cost of edges.

First, ACE deviates from A^* in the costs used for the computation of heuristic h values. In A^*, these are the edge (action) cost values. ACE only has access to cost bounds instead of exact costs, so the h-values are computed based on lower bounds. Specifically, the *loosest* (lowest) lower bound for each edge is utilized, to preserve heuristic consistency (a straightforward minor result; see Lemma 2 in the appendix).

The second deviation of ACE from A^* is that instead of simply using $c(e)$ to calculate $g(s)$ for a node s, an estimation loop is introduced (lines 13–18) to acquire estimated cost bounds. For the edge e connecting nodes n and s ($e = (n, s)$, line 13), the loop iterates over possible estimators until the bound $\mathcal{B}$ is met by η of the path ending in node s, or an alternative path with lower $g_{min}(s)$ was already found, or no estimators are left. In each iteration, line 15 computes $\underline{c}$ (the *tightest* lower bound for e over all estimators applied so far), and $\overline{c}$ (similarly, the *tightest* upper bound). When the loop is done, if a path with lower $g_{min}(s)$ is found, then $g_{min}(s)$, $g_{max}(s)$ are updated (lines 19–20).

Every call to GetEstimator(e) returns the next estimator from the set $\Theta(e)$ which has yet to be applied, or $\emptyset$ when none are left. ACE's theoretical guarantees (below) assume nothing about the order of the estimators selected by GetEstimator. However, to save runtime, $\Theta(e)$ is ordered by increasing runtime. Thus, each time GetEstimator is invoked on the edge e, it returns the next estimator θ_e^i with the shortest runtime. This is the approach we took in the experiments.

Remark 4 *Note that an estimation-indifferent planning algorithm is easily derived from Alg. 6, by modifying it to only consider the tightest bounds.*

6.4.1 Analysis of ACE: Guarantees

We start by proving the completeness of ACE, which follows almost immediately from the structure of A^*, inherited by ACE.

Theorem 9 (Completeness) ACE *always terminates and returns a plan when one*

exists.

Proof sketch. First, the fact that ACE always terminates follows from the same reasoning that A^* always terminates, together with the fact that there is a finite number of estimators per edge, each with finite runtime. Second, in order to prove that ACE terminates with a plan, when one exists, we can again rely on the same reasoning as in the proof of A^*, where we only have to show that each new node encountered during the search is inserted into OPEN. Every new node (generated in line 9) triggers $g_{min}(s) \leftarrow \infty$ (line 11). In line 12 $\underline{g} \leftarrow g_{min}(n)$, namely $\underline{g}$ is initialized to some finite number. Then, in lines 14–18 $\underline{g}$ may be updated again, but it is guaranteed to remain a finite number, since every lower bound returned by an estimator (indicated as $\underline{c}$) is also finite. Thus, the condition in line 19 is met, and so s is necessarily inserted to OPEN (line 23). ■

Next, we show that ACE is sound, i.e., if it returns a plan then it must be correct. To do this, we rely on Lemma 5, stated as follows (see appendix for full proof).

Lemma 5 (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.*

Proof. As mentioned before, ACE uses heuristic values that are computed by h (which is a consistent parameterized heuristic) and based on the lower bound estimate given for each relevant edge e by the first estimator returned by $\mathsf{GetEstimator}(e)$. We obtain a (standard) consistent heuristic h_α, with α defined to be an edge cost function, where the cost of each edge e is taken to be the lower bound estimate of the first estimator returned by $\mathsf{GetEstimator}(e)$. During the search ACE potentially utilizes additional estimations (as it tries to meet the target $\mathcal{B}$). Denote by $\Phi \subseteq \Theta_\Sigma$ the set that includes exclusively all estimators invoked by ACE during the search, and further denote β as an edge cost function, where the cost of each edge e is taken to be the tightest (highest) lower bound estimate of all the estimators returned by $\mathsf{GetEstimator}(e)$ during the search. Since $\alpha(e) \leq \beta(e)$ for every edge e, Lemma 1 assures that h_α is consistent w.r.t. $\mathcal{G}_\beta$ (where its nodes and edges are defined by the digraph that

corresponds to the planning problem $\mathcal{P}$). Then, following the main proof arguments of A^*'s optimality, consistency of h_α implies non-decreasing f-values along any path, thus whenever ACE selects a node for expansion, an optimal path (w.r.t. β) to that node has been found. Hence, the first plan π found by ACE has plan lower bound $c^{\Phi}_{min}(\pi)$ equal to the optimal plan lower bound w.r.t. Φ, i.e., $c^{\Phi}_{min}(\pi) = c^*_{\Phi}$, implying that π is an optimal optimistic plan w.r.t. Φ. ■

In P-MACE, soundness needs to be defined in terms of the bound requested of the solution. We define this property as follows. An algorithm is said to be $\mathcal{B}$-**sound** if every time it returns a plan reported as $\mathcal{B}$-optimal, the plan returned is indeed $\mathcal{B}$-optimal. Thm. 10 shows this is true of ACE.

Theorem 10 ($\mathcal{B}$-Soundness) *Provided with a consistent heuristic h,* ACE *is $\mathcal{B}$-sound.*

Proof. By induction on the order of nodes entering OPEN, starting from the base case of the start node, we show that each node n in OPEN satisfies $\eta(n) := g_{max}(n)/g_{min}(n)$ (where the case of $g_{min}(n) = 0$ is considered $\eta(n) = 1$, as there is no uncertainty). Thus, if a plan π is found, terminating at s_g, it necessarily means that $\eta(s_g) = \eta(\pi)$ is returned (i.e., the η returned is correct). In addition, according to Lemma 5, using a consistent h ensures that π is guaranteed to be an optimal optimistic plan w.r.t. some valid $\Phi \subseteq \Theta_\Sigma$. Hence, if $\eta(\pi) \leq \mathcal{B}$ is satisfied, then relying on Thm. 8, π is a $\mathcal{B}$-optimal plan, whereas in case it is not satisfied, then π is not guaranteed by ACE to be $\mathcal{B}$-optimal. ■

We now transition to discuss the optimality of ACE. To do this, we first clarify the following.

$\mathcal{B}$-Optimality: An algorithm is said to be *$\mathcal{B}$-optimal* if it is guaranteed to find a plan that can shown to be a $\mathcal{B}$-optimal plan (Def. 7) using estimators in Θ.

In the general case ACE is not $\mathcal{B}$-optimal. Not every plan returned will meet the target bound. It might even be that in generating the resulting plan, not all estimation options are exhausted, thus ACE could return a plan that is not $\mathcal{B}$-optimal, though one exists.

However, under some conditions, it is $\mathcal{B}$-optimal (Thm. 11). Intuitively, if every edge can be estimated to the desired degree of certainty, then starting from a bound on the path cost uncertainty which is under the threshold, and adding only edges with "good enough" cost estimates, it is possible to retain all paths explored with accumulated η lower or equal to $\mathcal{B}$. Then, the first solution found (if it exists) necessarily meets the requirement.

Theorem 11 (Special $\mathcal{B}$-Optimality) *Given a* P-MACE *problem $\mathcal{P}$ with suboptimality target $\mathcal{B}$, if every edge $e \in \mathcal{E}$ satisfies $c^i_{max}(e)/c^i_{min}(e) \leq \mathcal{B}$ (or $c^i_{min}(e) = c(e) = 0$) for some i (that can be different for each edge), and a consistent heuristic h is used, then* ACE *is $\mathcal{B}$-optimal.*

Proof. First, that fact that ACE is complete follows from Thm. 9. Second, similar to the proof of Thm. 10, and using the fact that for every edge e the bound $c^i_{max}(e)/c^i_{min}(e) \leq \mathcal{B}$ (or the equality $c^i_{min}(e) = c(e) = 0$) can be achieved (for some i), it can be shown by induction that each node n in OPEN satisfies $\eta(n) = g_{max}(n)/g_{min}(n) \leq \mathcal{B}$ (where again, $g_{min}(n) = 0$ is considered as $\eta(n) = 1$). Therefore, if a $\mathcal{B}$-optimal plan exists, then ACE will necessarily return such one. ■

It follows from Thm. 11 that ACE **is $\mathcal{B}$-optimal in the case of classical planning**, where an exact-cost estimate is available for every edge. See below.

Corollary 3 *If the cost of every edge can ultimately be estimated exactly, i.e., $\forall e \in \mathcal{E}$ it holds that $\exists i : c^i_{max}(e) = c^i_{min}(e)$, then necessarily $c^i_{max}(e)/c^i_{min}(e) = 1 \leq \mathcal{B}$, i.e.,* ACE *is $\mathcal{B}$-optimal (Thm. 11).*

6.4.2 Post-Search Improvements

We provide a simple post-search procedure (Proc. 7) that aims to reduce the cost uncertainty $\eta(\pi)$ when ACE terminates with $\eta(\pi) > \mathcal{B}$. We observe that even then, there may still be unused estimators for edges (ground actions) along the path (plan) found. It therefore may still be possible to improve $\eta(\pi)$ by applying additional estimators for edges that are already in π.

The basic idea is to loop over all the edges along the path corresponding to the

plan found, and for each one try to reduce its uncertainty by applying unused estimators. We emphasize that the tightest lower bound we may use to calculate $\eta(\pi)$, so that π can still be guaranteed to be optimal optimistic w.r.t. some Φ (which is needed in order to relate $\eta(\pi)$ to $\mathcal{B}$, due to Thm. 8), cannot be greater than the one already found. The reason is that a different plan $\tilde{\pi}$ might exist with $c_{min}(\tilde{\pi})$ equal to that f-value. For this reason only upper bounds are updated in the procedure.

Procedure 7 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
  θ ← GetEstimator(e)
  while η > B and θ ≠ ∅ do
    c, c̄ ← apply(θ)
    update c_max(π) using c̄
    η ← c_max(π) / c_min(π)
    θ ← GetEstimator(e)
  if η ≤ B then
    return η
return η
```

6.5 Empirical Evaluation

The theoretical properties of ACE give little insight as to its runtime behavior and success rate. ESE does not carry theoretical guarantees. We therefore focus on extensive experiments to evaluate ACE and ESE in a variety of benchmark planning problems where we control the need for estimation and target sub-optimality bound.

We developed *PlanDEM* (Planning with Dynamically Estimated Action Models), a planner that provides a concrete implementation of ACE and ESE, in C++. PlanDEM (available in Weiss and Kaminka (2023)) modifies and extends Fast Downward (FD) Helmert (2006) (v20.06), and inherits its use of the h_{max} heuristic Bonet and Geffner (2001), used in the experiments (where h_{max} was chosen

due to its consistency). The underlying FD search algorithm and data structures were modified appropriately. All experiments were run on an Intel i7-6600U CPU (2.6GHz), with 16GB of RAM, in Linux.

We modified standard planning problems from past planning competitions, to include synthetic estimators at several levels of uncertainty. Specifically, we selected 20 domains & problems with action costs Fox and Long (2003). For each, we generated a random variant with a target $\mathcal{B}$, and varying the number of actions with estimated (rather than precise) costs with a probability p_1. When we set $p_1 = 1$, the costs of all ground actions (edges in $\mathcal{G}_\Sigma$) are estimated by the planner. When $p_1 = 0$, all costs are given precisely. When $0 < p_1 < 1$, only some ground actions require estimation of the cost. Thus a pair $\mathcal{B}, p_1$ creates a new instance of a benchmark problem. The costs, their bounds and names of domains & problems used are described in Subsections 6.5.1 and 6.5.2.

Each estimated-cost edge e with precise cost $c(e)$ is provided with a set of three estimators of $c(e)$, $\Theta(e) = \{\theta_e^1, \theta_e^2, \theta_e^3\}$. These have $c^i_{max}(e)/c^i_{min}(e)$ uncertainty ratios of 4, 2, and 1 (the correct value), resp.

Since the problems we address have a target bound $\mathcal{B}$, but no other solution ranking criteria, we choose to prefer time over smaller uncertainty as a secondary ranking criterion. Given two $\mathcal{B}$-optimal solutions π_1, π_2 with corresponding $\eta^1 < \eta^2$, but runtimes $t^1 > t^2$, we consider π_2 to be better.

6.5.1 Domains and Problems Used in the Experiments

For the empirical evaluation based on IPC problem domains, we used domains and problems that appeared in the international planning competitions of 2008, 2011, 2014 and 2018. For each domain we chose two problems (ranging from small to large scale), and for each of them we tested all configurations reported here (resulting in many different variants per original problem file). The full list of domains (problems) follows below, and the files can be retrieved from Seipp et al. (2016).

- elevators-opt08-strips (p04, p06)
- barman-opt11-strips (pfile01-003, pfile01-004)

- floortile-opt11-strips (opt-p05-009, opt-p06-011)
- sokoban-opt11-strips (p04, p07)
- transport-opt11-strips (p02, p04)
- woodworking-opt11-strips (p06, p12)
- tetris-opt14-strips (p03-4, p04-6)
- agricola-opt18-strips (p08, p10)
- caldera-split-opt18-adl (p05, p10)
- data-network-opt18-strips (p17, p20)

6.5.2 Costs and Estimators

We generally used the original cost depicted in the PDDL domain and problem files, denoted as $c_{PDDL}(e)$. However, PlanDEM supports solely positive integer action costs, as it extends FD and thus inherits its main software architecture and data structures. As a result, for problems that have $c(e) = 1$, it is not possible to bound an estimate from below. In these cases we changed the original cost so as to be able to experiment with estimators that provide a bound lower than the original cost.

Thus, in order to obtain the desired uncertainty ratios for each edge e, we used $c_{PDDL}(e)$ as a basis for the following transformation.

- If the edge cost was to be estimated (i.e., with probability p_1 as described above), then the list of estimators is given by $\Theta(e) = \{\theta_1^e, \theta_2^e, \theta_3^e\}$, where the estimators respectively return

$$
\begin{aligned}
c^1_{min}(e) &= 1 \times c_{PDDL}(e), c^1_{max}(e) = 4 \times c_{PDDL}(e), \\
c^2_{min}(e) &= 2 \times c_{PDDL}(e), c^2_{max}(e) = 4 \times c_{PDDL}(e), \\
c^3_{min}(e) &= 2 \times c_{PDDL}(e), c^3_{max}(e) = 2 \times c_{PDDL}(e).
\end{aligned}
$$

  In this case, the true cost is taken to be $c(e) = 2 \times c_{PDDL}(e)$.

- Otherwise (true cost known and not estimated): $\Theta(e) = \{\theta_1^e\}$, where θ_1^e

returns

$$c^1_{max}(e) = c^1_{min}(e) = c_{PDDL}(e).$$

Hence, in this case $c(e) = c_{PDDL}(e)$.

6.5.3 Evaluation of ACE

We take p_1 to be one of $\{0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1\}$ and $\mathcal{B}$ values in the range $[1, 4]$ in jumps of 0.25, resulting in a total of 1820 runs. Note that $\mathcal{B} = 1$ requires precise cost, while $\mathcal{B} = 4$ is equivalent in our setting to having no requirement at all on precision (as the highest uncertainty ratio given by an estimator is 4). As one of the estimators has uncertainty ratio of 1, the cost of each edge can be eventually estimated perfectly, and thus Thm. 11 holds (i.e., the achievable η is 1). Therefore, ACE will always terminate with a $\mathcal{B}$-optimal solution.

Comparison to Baseline.

As the planning problem defined in this chapter is new, there is no immediate baseline for comparison other than the naive *estimation-indifferent* planning utilizing all available estimators.

We empirically contrast PlanDEM with estimation-indifferent planning. We begin by examining ACE's runtime behavior from the perspective of estimators utilization. In particular, as every edge considered has to be estimated at least once, we focus on the number of expensive estimators, i.e., (those with tighter ratios 2, 1).

Fig. 6.1 plots the ratio between the actual number of expensive estimations that ACE used and the maximum number of potential expensive estimations on the vertical axis, against the target $\mathcal{B}$ bound. Thus a lower ratio indicates an improved result, i.e., less expensive estimations used. The estimation-indifferent planning approach always uses all estimators, and therefore its ratio is always 1, indicated by the the straight solid horizontal black line at the top of the figure. The figure shows different curves for the several p_1 values. Each point on each curve averages 20 planning runs. The curves of $p_1 = \{0.01, 0.05\}$ were left out for clarity.

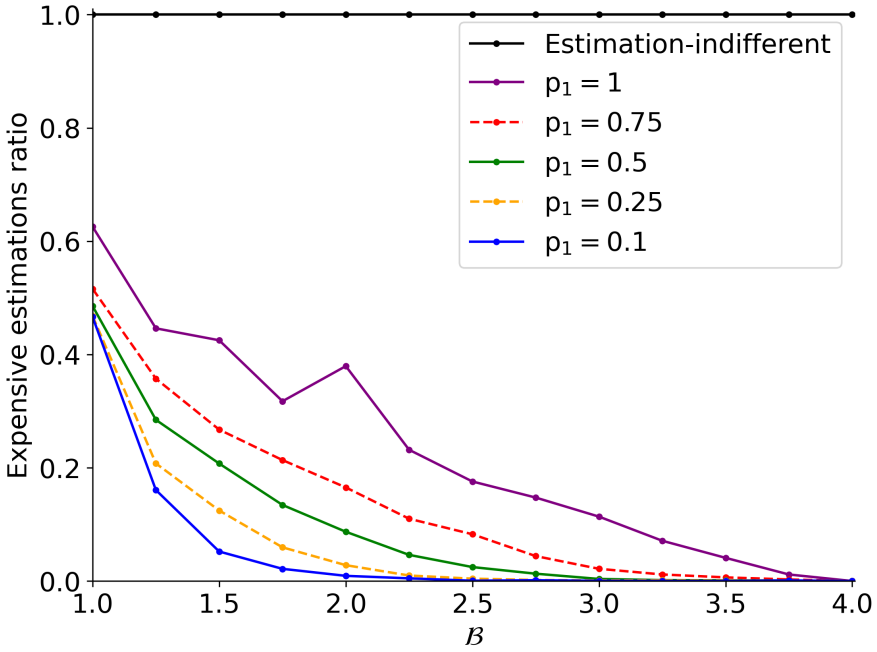


Figure 6.1: Normalized expensive estimations versus target bound $\mathcal{B}$. The curves from the bottom (blue, solid), to the top (purple, solid) correspond to the p_1 values $0.1, 0.25, 0.5, 0.75$ and 1. Success rate is 100% for all runs. The uppermost curve (black, solid) is the baseline estimation-indifferent

We first compare the estimation-indifferent approach with ACE. Consider the worst-case where the cost of *all* actions is estimated ($p_1 = 1$, solid purple), and the target bound requires maximum accuracy ($\mathcal{B} = 1$). Here, ACE utilizes only 62% of the expensive estimators. For lower p_1, it improves up to 46% ($p_1 = 0.1$). This is due to the condition on $\underline{g}$ in line 14 of ACE, which avoids unnecessary estimations in case another path is examined that leads to a node that is already in OPEN and has lower f-value. Without sacrificing accuracy, ACE is superior to its basic competitor that uses full estimation (indicated as the straight line of ratio 1).

Next, the generally decreasing trend of the curves in Fig. 6.1 suggests that substantial savings may be achieved even by minor relaxation of the uncertainty bound (allowing greater $\mathcal{B}$). Furthermore, as p_1 decreases the curves drop rapidly, allowing ACE to further increase its efficiency.

Runtime and Quality.

Figure 6.2 shows actual and approximate planning runtime (measured in CPU seconds), on the vertical axis (logarithmic scale). As the time taken by expensive estimators is arbitrary in these experiments, we wanted to get a feel for the planning runtime under various assumptions of computation time. The bottom curve in the figure is the actual runtime of PlanDEM without the estimation. The other curves, bottom-to-top, show what the runtime would be had we added the estimation runtime (number of expensive estimates multiplied by assumed per-estimate runtime). This emphasizes the importance of avoiding unnecessary estimations, which dominate the total runtime when estimation time is high (compared to the heuristic estimation time, contained within the baseline). It also strengthens the argument mentioned above that significant potential savings can be obtained by even slightly loosening the uncertainty bound.

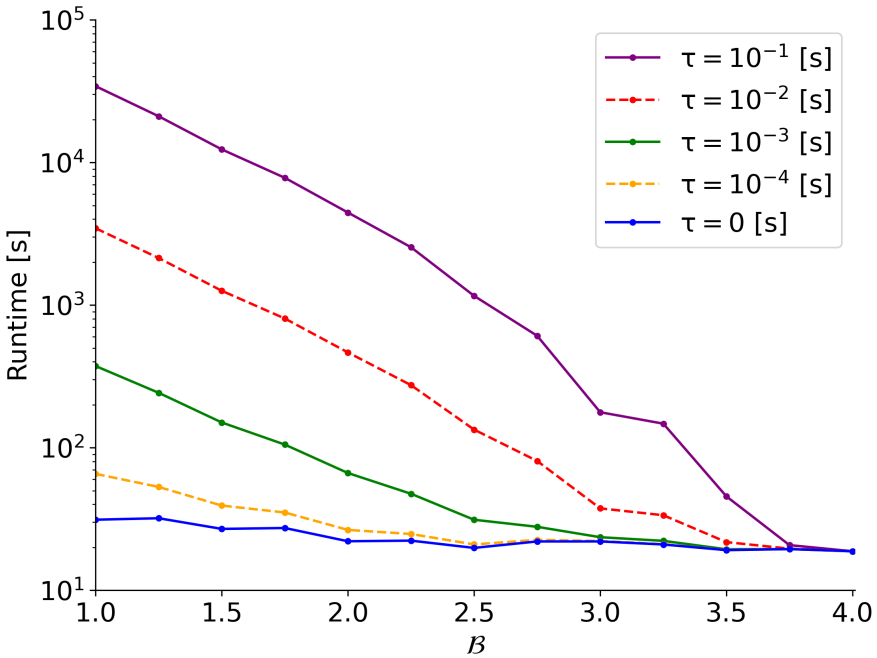


Figure 6.2: PlanDEM runtime in CPU seconds. Bottom (blue, solid) to top (purple, solid) curves correspond to per-estimation times of 0, 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1} seconds for the expensive estimators. $p_1 = 0.5$ for all cases.

We now move to examine the quality of the solutions produced by ACE. Figure 6.3 shows mean η values achieved for different target $\mathcal{B}$ bounds. Recall that we consider shorter planning time to be better than achieving higher accuracy. Hence, the fact that η tends to $\mathcal{B}$ when p_1 is high indicates that ACE is efficient w.r.t. usage of expensive estimations. When p_1 values are low, most costs are precisely known, and overall uncertainty should be low even with a small amount of expensive estimations (see Fig. 6.1). Indeed, η values are well below the target bounds.

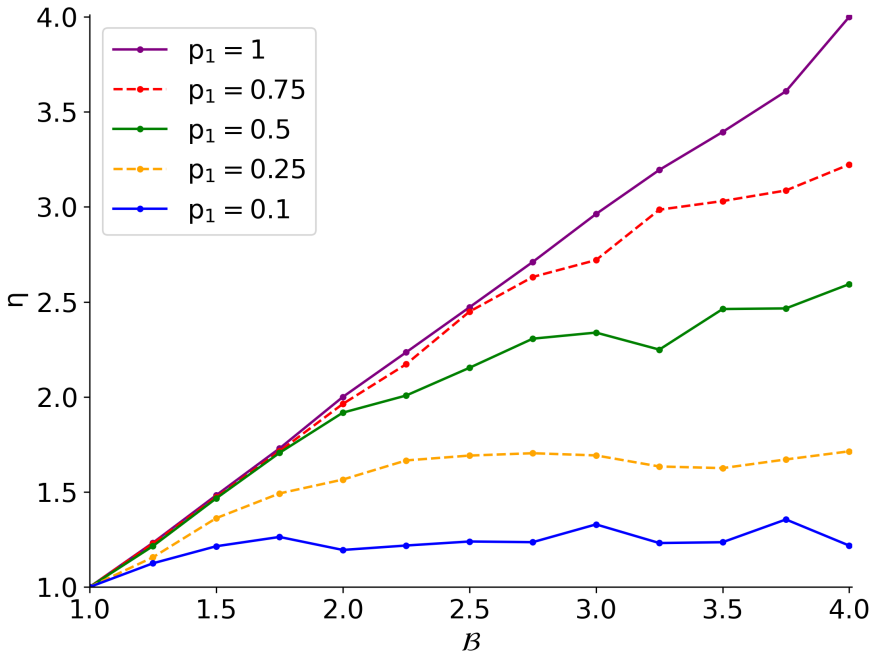


Figure 6.3: Mean η vs $\mathcal{B}$ target bounds. The curves bottom (blue, solid) to top (purple, solid) correspond to the p_1 values $0.1, 0.25, 0.5, 0.75$ and 1.

6.5.4 Evaluation of ESE

The section above examines ACE when it can always succeed in finding a $\mathcal{B}$-solution, and thus ESE is never invoked. This section examines cases where it might not. We introduce probabilities for the existence of the second and third estimators, denoted by p_2 and p_3. When setting either p_2 or p_3 to values different than 1, the achievable η increases, as tighter estimates are no longer

available. This creates cases where ACE can fail, and ESE might be applicable, and will be invoked. For our experiment we used $p_1, p_2 \in \{0.25, 0.5, 0.75, 1\}$, $p_3 \in \{0.25, 0.5, 0.75\}$ and $\mathcal{B} \in \{1.5, 2, 2.5, 3, 3.5\}$. This resulted in 4800 planning problem variants to be solved.

The results are summarized in Table 6. Each row corresponds to a different $\mathcal{B}$ target, and summarizes 960 problems. Left to right, the columns show the number of problems (in parentheses, percentage of problems) in which ACE succeeded, ESE was applicable and invoked, ESE was successful, the resulting η and usage of costly estimations for both ACE and ESE. The table points to several insights.

$\mathcal{B}$	ACE Success (%)	ESE Invoked (%)	ESE Success (%)	η ACE (ESE)	Costly Estimations ACE (ESE)
1.5	276 (28.75)	174 (18.13)	50 (28.74)	2.23 (1.98)	132566045 (511)
2	548 (57.08)	171 (17.81)	59 (34.50)	2.63 (2.14)	145826527 (602)
2.5	701 (73.02)	145 (15.10)	52 (35.86)	2.95 (2.33)	136260264 (544)
3	806 (83.96)	97 (10.10)	55 (56.70)	3.29 (2.51)	93636149 (382)
3.5	897 (93.44)	38 (3.96)	23 (60.53)	3.67 (2.68)	29996783 (144)

Table 6: Summarized data of ACE & ESE experiments (960 runs per $\mathcal{B}$ value). The values in column 5 are averages of η over all instances where ESE took place, and the values in column 6 are sums of costly estimations over all those instances.

First, the number of ESE invocations is not equal to the number of ACE failures. There are cases where ACE exhausted all estimation options for the actions in the plan and thus ESE is not applicable. As the success ratio of ACE increases with $\mathcal{B}$, the number of ESE invocations decreases (as this procedure is required less frequently).

Second, a key insight is that ESE has a considerable success ratio, which increases with $\mathcal{B}$. This is striking, given that it utilizes only a small amount of estimates w.r.t. the amount ACE uses, i.e., its affect on the total runtime is minor.

To explain this result, we examine the change in η due to ESE. The data in column 5 from the table indicates values averaged over both successful and failed runs (when the bound is not met), without normalization. We therefore

examine the normalized relative change in η, defined as

$$\eta_{rel} := (\eta^{\mathsf{ESE}} - \eta^{\mathsf{ACE}})/(\eta^{\mathsf{ACE}} - 1),$$

where the value 1 in the denominator, which is the best (lowest) achievable η, serves as to normalize the change.

The mean normalized relative change is only -5.72%. This implies that ACE typically comes very close to the target (up to roughly several percent). It then becomes relatively easy for ESE to use a few more estimates, to drive η below the target.

6.5.5 An ACE Experiment on Real-World Data

Inspired by the contrast between the simplistic estimates of the Transport Sequential domain and the commercial-grade estimation carried out by TruckNet (TruckNet (2021); see introduction), we created a package delivery problem based on real data for the Ontario province in Canada. We first provide a brief description in high-level, and then give the full details in Subsection 6.5.5 below.

The problem is deceptively small: 9 cities, with 30 road segments (distances from Google Maps), 2 trucks each with 4-package capacity, and 6 packages to be delivered (spread across different initial and terminal locations). There are three actions: drive, pickup and drop. The objective is to minimize monetary costs (a function of both distance and time), subject to the inherent uncertainty of travel times.

The cost per distance was computed online based on the truck model depreciation, financing, licensing, insurance, and fuel prices for the region. The cost per minute was computed from reported driver salaries in Canada. Two estimators were used for the drive action cost: the first (cheap) assumed the speed would be between heavy traffic and no traffic conditions; the second (more costly) estimator used pessimistic and optimistic times reported by Google Maps for a specific day and hour. Both estimators yield travel time in minutes, which was then used with the cost-per-minute and cost-per-distance. In order to test a more computationally intensive scenario, we assumed that the costs are state-

dependent, which significantly increases the required time for overall estimation.

Ontario Transportation Logistics Example

We now fully describe the details of the experiment. The road system considers 9 cities with 30 road segments connecting them, where each road length is specified using a 100 meter resolution, where the data is taken from Google Maps. The cost function per action a depends on the distance traveled d and on time duration t, and is given by $c(a) = c_1 \times d + c_2 \times t$, where $c_1 = 0.56$ [USD/km] and $c_2 = 0.5$ [USD/minute] are the cost-per-km and cost-per-minute coefficients. c_1 was taken from Vincentric (2022) and is based on data tailored to a specific set of assumptions: Ford Transit 150, 2021 model, with base crew medium sized roof slide 148WB AWD trim, commuting 30000 km annually, in Ontario province. It calculates an average total cost-per-km based on the above specifications, considering current fuel prices, depreciation and maintenance costs, license and registration fees, insurance costs and even financing. c_2 is based on an estimate of a truck driver employer's cost in Canada (see e.g., data from Glassdoor).

The domain file is similar to the aforementioned "Transport Sequential", having three action templates: drive, pick-up and drop, where the prices of the two latter actions are assumed to be exact and are based on a 20 minutes time duration. Each drive action has two estimators: the first, which doesn't incur additional runtime, assumes nothing about the specific road segment and thus uses conservative estimates of 20 km/hour and 100 km/hour for unfavorable and favorable road conditions, respectively; and the second uses pessimistic and optimistic travel time estimates taken from Google Maps by manual queries for a specific day and hour (Friday, 10 AM) and the fastest route. All the required data for reproducing the experiment is contained in 3 files, corresponding to the domain, problem and estimators, that appear in Weiss and Kaminka (2023).

We evaluated ACE on the problem for $\mathcal{B} \in \{1, 1.1..., 2.4\}$, and the results are summarized in Table 7. As can be seen, the results are in full correspondence with those obtained by experimenting with the modified IPC benchmarks.

We highlight several interesting conclusions from the experiment:

- ACE saved between 60% and ∼100% costly estimations compared to estimation-indifferent planning, while maintaining a 100% success rate on feasible requirements (as $\mathcal{B} = 1, 1.1$ turned out to be out of reach).
- The best attainable estimates had uncertainty ratios ranging from 8.5% to 53.8%, demonstrating that cost uncertainty is oftentimes unavoidable. In these cases standard planning would generate inferior plans.
- Assuming a 1ms query time for travel time prediction (representing access time to a local database), and even in the worst result of only 60% savings, planning time is 43 minutes instead of 107 minutes for estimation-indifferent planning. For query times of 10ms (representing a very fast access to an online database), this translates to 7 hours of planning time instead of 18 hours.

We point out that this experiment was based on a precompiled file containing the required estimation data, so that during planning cost estimates were obtained by repeated access to the file, with many ground actions having the same cost estimates. In such cases where multiple actions (or edges) have the same estimate, using a cache mechanism, in order to reduce calls to external modules, should significantly improve the results. The design and implementation of an effective cache is left for future research.

6.6 Related Work

ACE's repeated evaluation and selection between cost estimators *complements* the process of choosing between multiple heuristics during planning Karpas et al. (2018). Heuristics are applied to states (vertices in the state-space graph), while costs apply to actions (edges).

In shortest-path search, cost uncertainty is considered by assuming explicit or implicit knowledge about edge cost distributions, and then solved by performing calculations involving the full graph, typically minimizing expectation Kwon et al. (2013); Shahabi et al. (2015). ACE makes no such assumptions.

$\mathcal{B}$	Success	η	θ_2 Used	θ_2 Saved	θ_2 Usage (%)	T_0[s]	T_1[s]	Saved T_1[s]	T_{10}[s]	Saved T_{10}[s]
1	0	1.18	2537169	3858085	39.67	54.12	2591.29	3858.09	25425.81	38580.85
1.1	0	1.18	2537169	3858085	39.67	61.89	2599.06	3858.09	25433.58	38580.85
1.2	1	1.18	2565641	4254910	37.62	66.16	2631.80	4254.91	25722.57	42549.10
1.3	1	1.28	2653524	7091775	27.23	93.93	2747.46	7091.78	26629.17	70917.75
1.4	1	1.39	2469987	8222062	23.10	101.26	2571.25	8222.06	24801.13	82220.62
1.5	1	1.47	2253078	8587468	20.78	102.10	2355.18	8587.47	22632.88	85874.68
1.6	1	1.58	2047551	8909325	18.69	106.90	2154.45	8909.33	20582.41	89093.25
1.7	1	1.66	1809278	8843618	16.98	105.22	1914.50	8843.62	18198.00	88436.18
1.8	1	1.76	1508847	9375424	13.86	94.54	1603.39	9375.42	15183.01	93754.24
1.9	1	1.88	1149769	8486273	11.93	72.78	1222.55	8486.27	11570.47	84862.73
2	1	1.96	722353	8102761	8.19	84.37	806.72	8102.76	7307.90	81027.61
2.1	1	2.03	237430	6863791	3.34	69.62	307.05	6863.79	2443.92	68637.91
2.2	1	2.05	733	6349392	0.01	60.23	60.96	6349.39	67.56	63493.92
2.3	1	2.16	31	6350040	5×10^{-4}	61.55	61.58	6350.04	61.86	63500.40
2.4	1	2.24	7	6350479	10^{-4}	60.59	60.59	6350.48	60.66	63504.79

Table 7: Summarized data of ACE experiments with the Ontario Transportation Logistics problem. Notations: θ_2 denotes the costly estimations, and T_0, T_1, T_{10} correspond to planning times using $\tau_2 = 0, 1, 10$ ms, respectively. Usage and savings are w.r.t. to the results of estimation-indifferent planning.

Related efforts tackle graph search in the context of motion planning, where verification of edge existence is expensive Narayanan and Likhachev (2017); Mandalika et al. (2019). The solution approach dynamically decides when to search and when to evaluate edges, so as to minimize overall planning time, but is limited to existence verification. This is a special case of general action cost computation in P-MACE (when the cost tends to infinity, an edge can be considered non-existent). Existence verification is carried out only once per selected edge, whereas P-MACE may require multiple estimations for the same edge.

A closer line of work addresses shortest path problems where obtaining the true edge cost is considered to be computationally expensive. Dellin and Srinivasa (2016) present a framework for lazy search w.r.t. expensive edge evaluations. They assume two estimates are provided for each edge: a computationally-cheap lower-bound (no upper bound), and a computationally-expensive exact cost. Clearly this is a special class of P-MACE problems. The lazy search cannot solve P-MACE problems in general, because it does not admit upper bound estimates, and cannot target $\mathcal{B}$-bounded solutions. Technically, the lazy search algorithm strictly requires two estimates, of which one is the exact cost. This requirement does not exist for ACE.

We emphasize that P-MACE problems can, in general, be solved *without* utilizing all the most expensive estimators in the solution path, hence it is insufficient to simply be lazy w.r.t. to the most expensive estimators.. For instance, ACE can sometimes solve a P-MACE problem while utilizing *all* estimates for some edges in the solution path, some of the estimates for other edges in the path, and only one estimate for the remaining edges.

6.7 Discussion

ACE's theoretical properties hold regardless of the order of the estimators given by GetEstimator. However, even *partial ordering* of the estimators by expected run-time can be used by GetEstimator to return cheaper estimators first, making ACE faster in practice.

Typically, algorithms that try to reduce resource consumption of one type, have to make the sacrifice of consuming more resources of a different type (the classical trade-off being run-time vs. memory). ACE is able to greatly reduce the number of expensive estimators utilized, compared to the estimation-indifferent approach (as demonstrated in Fig. 6.1). Since the estimation-indifferent baseline considered here has the same node-expansion behavior of standard A^* (when it considers only the tightest lower bounds as costs), it is guaranteed to be optimally efficient (when used with a consistent heuristic).

One may therefore expect that ACE should be inefficient w.r.t. the number of node expansions. But this is not true. In fact, experiments suggest that ACE uses roughly the same number of node expansions as the estimation-indifferent baseline. We believe the trade-off is elsewhere. ACE uses a *best-effort approach*, and so it is not guaranteed to return a $\mathcal{B}$-optimal plan in the general case, but only in the special case where an exact-cost estimate is given for every action. We expect that future extension of ACE that guarantee general $\mathcal{B}$-optimality will need to increase search effort (i.e., will expand a larger number of nodes).

An assumption made in this chapter is that estimators represent cost uncertainty by an interval composed of lower and upper bounds. In cases where statistical estimators are used, statistical properties (e.g., standard deviation)

may be used instead as bounds, in which case the plan accuracy bound should be considered soft (or approximate) rather than strict.

6.8 Conclusion

We introduced *planning with multiple action cost estimators* (P-MACE), a generalization of deterministic planning that allows the domain expert to provide the planner with multiple cost estimators, each with its own uncertainty and computational requirements. During planning, the planner is then responsible for balancing the computational cost of estimating action costs, and the accuracy of the estimates. We then introduced ACE—a generalization of A^*—and showed that it is optimal in case each action cost can ultimately be estimated exactly (and also in other cases). To improve upon it when it fails to meet the target, we introduced a post-search procedure that increases ACE's success ratio. Extensive experiments show the algorithms are very effective.

ACE is the first attempt to solve P-MACE problems as introduced here. We plan to focus future algorithmic improvements on general $\mathcal{B}$-optimality. Another important future direction is to improve the selection of estimators so as to minimize their use, by considering meta-information, such as predicted values for c_{min}, c_{max} and/or run-time prior to the actual application of an estimator.

Chapter 7

Discussion and Future Work

"All models are wrong, but some are useful."

George Edward Pelham Box

7.1 Discussion

The central premise of this dissertation is that action or edge costs in planning and shortest-path problems are not always immediately available, nor are they necessarily cheap to compute. Instead, costs may need to be obtained through a series of increasingly expensive estimation procedures, each offering tighter bounds on the true value. This framework generalizes classical planning and search formulations and provides a natural way to reason about computation as a resource. While powerful, this shift introduces assumptions, limitations, and trade-offs that are worth discussing.

A first assumption underlying all formulations is that estimators return valid and monotonically tightening bounds on the true cost. This is a strong requirement: it ensures theoretical guarantees of correctness, but in practice, learned or data-driven estimators may violate monotonicity due to noise or model drift. The guarantees presented rely on these properties being maintained, meaning that robustness to imperfect estimators remains a challenge. Another assump-

tion stems from implementation constraints: in adapting our algorithms to standard planning frameworks such as Fast Downward, costs were restricted to positive integers. This sometimes required transforming problem definitions, which may reduce fidelity to original domains.

Several limitations emerge from the chosen approach. First, many of the experiments employed synthetic estimators, generated by scaling or bounding existing costs. While this enabled controlled evaluations, it does not fully capture the complexity and correlations of real-world estimators. The Ontario logistics experiment offered a more realistic demonstration, but it too relied on precompiled data rather than live estimation queries. This gap highlights that while the algorithms are general, their performance in truly dynamic environments with noisy or adaptive estimators has not yet been fully explored. Second, although our formulations accommodate uncertainty explicitly, they remain tied to deterministic bounds, and thus exclude stochastic or probabilistic estimation models that are common in practice. The approach can still be utilized without change for bounds that are derived from expected values and confidence intervals instead of deterministic values, but this renders the associated theoretical guarantees irrelevant.

Another important dimension concerns trade-offs between search effort and estimation effort. The algorithms introduced in this dissertation, such as BEAUTY, A-BEAUTY, BEAST, and A-BEAUTY&BEAST, explicitly exchange additional node expansions for reductions in expensive estimator calls. For example, A-BEAUTY often required 1.8× to 8.5× more expansions than estimation-indifferent baselines but saved up to 55% of costly estimations. Similarly, BEAST reduced the number of expensive estimators by an average of 40% compared to uninformed baselines. These results underscore the value of modeling estimator time explicitly, but they also make clear that the savings depend heavily on the distribution and accuracy of available estimators. In cases where cheap estimators are already accurate, the benefits are substantial; in other cases, overhead from additional expansions may dominate.

Practical considerations also arise in terms of integration into existing planning systems. While PlanDEM demonstrated that multi-estimator planning could

be built on top of Fast Downward with relatively modest architectural changes, performance in real applications will likely depend on additional systems-level techniques. For instance, caching repeated estimator calls in domains where many actions share identical estimates was not implemented in our evaluation, but could significantly reduce overhead. Moreover, operational knobs such as thresholds for pruning or stopping criteria in anytime variants introduce tunable parameters whose effectiveness depends on domain-specific characteristics.

Finally, the scope of evaluation reflects both strengths and limitations. Experiments across IPC benchmark domains confirmed the generality of the approach, and the Ontario case study highlighted practical relevance. Yet, the domains largely stem from classical AI planning benchmarks, and further validation in robotics, logistics, or other large-scale decision-making systems remains to be done. In summary, the proposed framework broadens the theoretical foundation of planning and shortest-path search, but its practical impact ultimately hinges on the behavior of real-world estimators, system integration issues, and the willingness of practitioners to accept expanded search as a trade-off for reduced estimation cost.

7.2 Future Work

This dissertation has introduced a framework for planning and shortest-path search in graphs with multiple cost estimators, together with algorithmic contributions that exploit the trade-off between estimator accuracy and computational cost. While these contributions advance both theory and practice, they also open up a number of promising avenues for future research.

On the algorithmic front, this dissertation has already taken steps toward integrating heuristic search into the multi-estimator framework. Specifically, we showed that heuristic functions can be safely computed using the loosest lower-bound estimates of edge costs, ensuring that heuristic evaluations remain cheap and that the property of consistency is preserved. While effective, this represents only a baseline approach. More sophisticated strategies could leverage

tighter edge cost estimates selectively, for example on edges that are heavily reused across candidate paths, where improved accuracy may yield significant guidance benefits. Another promising idea is to design schemes that explicitly couple the use of cost estimates in both g-value and h-value calculations, potentially supported by caching mechanisms that share estimation results across these two components of search. Such extensions could bring the advantages of heuristics more fully into the multi-estimator setting, achieving better trade-offs between search effort, estimation effort, and solution quality.

Another concrete line of work lies in expanding the range of estimator models supported. The current framework assumes deterministic bounds that monotonically tighten, yet many real-world settings involve stochastic models, data-driven predictions, or simulators whose outputs are not guaranteed to behave monotonically. Extending the theory to probabilistic bounds, or to settings where estimator reliability must be learned online, would increase robustness and broaden applicability. Similarly, the algorithms could be adapted to domains where cost information is only partially observable, or where estimates interact with other forms of uncertainty, such as stochastic action outcomes.

Beyond algorithmic enhancements, future research should further explore integration into real-world planning and robotics systems. The results presented here already point to substantial potential savings in domains where estimator time is costly, such as transportation logistics and robotics motion planning. However, large-scale deployment requires addressing systems-level considerations: caching, parallelization, distributed estimator calls, and interaction with external services. Exploring these aspects would transform the framework from a theoretical tool into a practical component of modern planning architectures.

More broadly, the line of work introduced in this dissertation suggests a shift in how planning and search are conceptualized. Rather than treating estimation as an external preprocessing step or as a negligible overhead, estimation becomes a first-class citizen in the problem definition. This perspective resonates with ongoing trends in AI and robotics, where algorithms must increasingly balance accuracy, computation, and uncertainty. Extending the framework to encompass richer cost models, multi-agent settings, or hierarchical estimation

pipelines could further deepen its theoretical and practical significance.

In summary, while the contributions of this dissertation provide a solid foundation for reasoning about multiple cost estimators in planning and search, the field remains wide open. Progress will depend both on technical extensions—ranging from heuristic design to stochastic models—and on broader integration with practical systems. Together, these directions promise to extend the impact of the framework, bringing it closer to the complex, uncertain, and resource-constrained decision-making tasks encountered in real-world applications.

7.3 Conclusion

This dissertation has re-examined the foundations of planning and shortest-path search through the lens of cost estimation. In contrast to the classical assumption that action or edge costs are known exactly and obtained at negligible expense, the work here has shown how planning and search can explicitly account for the time and uncertainty inherent in computing those costs. By formulating new problem variants and developing corresponding algorithms, the dissertation has established a principled framework in which estimation effort and search effort are jointly optimized.

The contributions span multiple levels. At the conceptual level, the idea of online modeling for offline planning challenged the prevailing separation between modeling and search, proposing that estimation itself should be integrated into the planning process. At the theoretical level, the generalized graph formulations with multiple estimators led to new problems such as the shortest-path tightest lower bound (SLB), the shortest-path tightest upper bound (SUB), and the tightest admissible shortest path (TASP). These problems extend classical definitions in meaningful ways, grounding the study of admissibility and optimality in contexts where costs are uncertain or expensive to obtain. At the algorithmic level, the dissertation introduced extensions and combinations of well-known algorithms and techniques—such as Uniform Cost Search, A*, Bounded-Cost A*, Anytime Weighted A*, Branch and Bound, and greedy search—to fit the new multi-estimator framework. This resulted in a family of algorithms,

including BEAUTY, A-BEAUTY, BEAST, BEAUTY&BEAST, and ACE, each tailored to exploit estimator bounds and computation time while retaining the theoretical guarantees of their classical counterparts. Finally, at the practical level, empirical evaluations on planning benchmarks and a real-world logistics case study demonstrated the viability of these ideas in realistic domains.

Taken together, these results establish a new perspective on search and planning: one in which the computational and informational cost of estimation is inseparable from the task of decision-making itself. This shift not only broadens the theoretical landscape of search but also aligns with the realities of modern AI applications, where uncertainty, external computation, and learned models are integral components of the problem.

The framework developed in this dissertation offers both a rigorous foundation and a set of practical tools for future research. While there remain many challenges ahead, the core message is clear: by treating estimation as a resource to be managed rather than ignored, we can design planning and search systems that are more efficient, more expressive, and ultimately more aligned with the complexities of the real world.

Bibliography

Cameron Allen, Michael Katz, Tim Klinger, George Konidaris, Matthew Riemer, and Gerald Tesauro. Efficient black-box planning using macro-actions with focused effects. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 2021.

Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. A review of learning planning action models. *The Knowledge Engineering Review*, 33, 2018.

Jorge A Baier, Christian Fritz, Meghyn Bienvenu, and Sheila A McIlraith. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI*, pages 1509–1512, 2008.

Jim Blythe. Decision-theoretic planning. *AI magazine*, 20(2):37–37, 1999.

Mark S Boddy. Imperfect match: Pddl 2.1 and real applications. *Journal of Artificial Intelligence Research*, 20:133–137, 2003.

Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

CW Chan and TY Kam. A procedure for power consumption estimation of multi-rotor unmanned aerial vehicle. In *Journal of Physics: Conference Series*, volume 1509, page 012015. IOP Publishing, 2020.

Amanda Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez, Carlos Linares López, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012.

Christopher Dellin and Siddhartha Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 26, pages 459–467, 2016.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Nineteenth International Conference on Automated Planning and Scheduling*, 2009.

Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*, pages 99–115, 2012.

Patrick Eyerich. *Beyond classical planning: temporal and probabilistic extensions*. PhD thesis, Dissertation, Universität Freiburg, 2013, 2013.

Ariel Felner. Position paper: Dijkstra's algorithm versus uniform cost search or a case against Dijkstra's algorithm. In *Proceedings of the International Symposium on Combinatorial Search*, volume 2, pages 47–51, 2011.

Maria Fox and Derek Long. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

Guillem Francès, Miquel Ramiŕez, Nir Lipovetzky, and Hector Geffner. Purely declarative action descriptions are overrated: Classical planning with simulators. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

Guillem Frances, Miquel Ramírez Jávega, Nir Lipovetzky, and Hector Geffner. Purely declarative action descriptions are overrated: Classical planning with simulators. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4294–4301, 2017.

Harary Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4): 583–599, 1969.

Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4: 265–293, 2021.

Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

Michael Gianfelice, Haitham Aboshosha, and Tarek Ghazal. Real-time wind predictions for safe drone flights in toronto. *Results in Engineering*, 15:100534, 2022.

Daniel Gnad, Alvaro Torralba, Martín Domínguez, Carlos Areces, and Facundo Bustos. Learning how to ground a plan–partial grounding in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7602–7609, 2019.

Peter Gregory, D. Long, M. Fox, and J. C. Beck. Planning modulo theories: Extending the planning paradigm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2012a.

Peter Gregory, Derek Long, Maria Fox, and J Christopher Beck. Planning modulo theories: Extending the planning paradigm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, pages 65–73, 2012b.

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

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

David Hutchinson, Anil Maheshwari, and Norbert Zeh. An external memory data structure for shortest path queries. *Discrete Applied Mathematics*, 126(1):55–82, March 2003. ISSN 0166-218X. doi: 10.1016/S0166-218X(02)00217-2. URL `https://www.sciencedirect.com/science/article/pii/S0166218X02002172`.

Shahid Jabbar. *External Memory Algorithms for State Space Exploration in Model Checking and Action Planning*. PhD Dissertation, Technical University of Dortmund, Dortmund, Germany, 2008.

Hosang Jung and Junsu Kim. Drone scheduling model for delivering small parcels to remote islands considering wind direction and speed. *Computers & Industrial Engineering*, 163:107784, 2022.

Subbarao Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1601. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

Erez Karpas, Oded Betzalel, Solomon Eyal Shimony, David Tolpin, and Ariel Felner. Rational deployment of multiple heuristics in optimal state-space search. *Artificial Intelligence*, 256:181–210, 2018.

S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning a^*. *AIJ*, 155(1–2): 93–146, 2004.

R.E. Korf. Minimizing disk I/O in two-bit breadth-first search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 23, pages 317–324, 2008a.

R.E. Korf. Comparing search algorithms using sorting and hashing on disk and in memory. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-16*, pages 610–616, 2016.

Richard E Korf. Linear-time disk-based implicit graph search. *Journal of the ACM (JACM)*, 55(6):1–40, 2008b.

Changhyun Kwon, Taehan Lee, and Paul Berglund. Robust shortest path problems with two uncertain multiplicative cost coefficients. *Naval Research Logistics (NRL)*, 60(5):375–394, 2013.

Steven M. LaValle. *Planning Algorithms*. Cambridge Univerity Press, 2006.

Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Aditya Mandalika, Oren Salzman, and Siddhartha Srinivasa. Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 476–484, 2018.

Aditya Mandalika, Sanjiban Choudhury, Oren Salzman, and Siddhartha Srinivasa. Generalized lazy search for robot motion planning: Interleaving search and edge evaluation via event-based toggles. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 745–753, 2019.

Thomas L McCluskey. Pddl: A language with a purpose? In *ICAPS*, 2003.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Venkatraman Narayanan and Maxim Likhachev. Heuristic search on graphs with existence priors for expensive-to-evaluate edges. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, pages 522–530, 2017.

Shinkoh Okada and Mitsuo Gen. Fuzzy shortest path problem. *Computers & Industrial Engineering*, 27(1-4):465–468, 1994.

PTV-Group. Truck ETA calculation with the PTV drive&arrive API. `https://www.myptv.com/en/logistics-software/ETA-software-ptv-driveandarrive`, 2023. Accessed: 2023-8-5.

Jussi Rintanen. Impact of modeling languages on the theory and practice in planning research. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Jendrik Seipp, Florian Pommerening, Silvan Sievers, and Guillem Francès. Fast downward benchmark collection. `https://github.com/aibasel/downward-benchmarks`, 2016. Accessed: 2023-03-14.

Mehrdad Shahabi, Avinash Unnikrishnan, and Stephen D Boyles. Robust optimization strategy for the shortest path problem under uncertain link travel cost distribution. *Computer-Aided Civil and Infrastructure Engineering*, 30(6): 433–448, 2015.

Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *ICRA*, pages 3310–3317, 1994.

Roni Stern, Ariel Felner, Jur van den Berg, Rami Puzis, Rajat Shah, and Ken Goldberg. Potential-based bounded-cost search and anytime non-parametric A*. *Artif. Intell.*, 214:1–25, 2014.

Bradley S. Stewart and Chelsea C. White III. Multiobjective A*. *Journal of the ACM (JACM)*, 38(4):775–814, 1991.

Nathan R Sturtevant and Jingwei Chen. External memory bidirectional search. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-16*, pages 676–682, 2016.

TruckNet. Trucknet enterprise: Revolutionising the world of freight management. Logistics Tech Outlooks: https://www.trucknet.io/wp-content/uploads/2021/10/629764_418641774086freight-management-europe-cover774086418641.pdf, 2021. Accessed: 2021-12-15.

Vincentric. Driving costs calculator. https://carcosts.caa.ca/, 2022. Accessed: 2023-03-14.

Jeffrey Scott Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, June 2001. ISSN 0360-0300. doi: 10.1145/384192.384193. URL https://doi.org/10.1145/384192.384193.

Eyal Weiss. A generalization of automated planning using dynamically estimated action models–dissertation abstract. In *32nd International Conference on Automated Planning and Scheduling*, pages 1–3, 2022.

Eyal Weiss and Gal A. Kaminka. Position paper: Online modeling for offline planning. In *Proceedings of the 1st ICAPS Workshop on Reliable Data-Driven Planning and Scheduling*, 2022.

Eyal Weiss and Gal A. Kaminka. PlanDEM. https://github.com/eyal-weiss/plandem-public, 2023. Accessed: 2023-07-26.

Eyal Weiss, Ariel Felner, and Gal A. Kaminka. A generalization of the shortest path problem to graphs with multiple edge-cost estimates. In *Proceedings of the European Conference on Artificial Intelligence*, volume 372, pages 2607–2614, 2023.

Eyal Weiss, Ariel Felner, and Gal A Kaminka. Tightest admissible shortest path. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 643–652, 2024.

Richard W Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial intelligence*, 13(1-2):133–170, 1980.

Hebrew Abstract