ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230567

A Generalization of the Shortest Path Problem to Graphs with Multiple Edge-Cost Estimates

Eyal Weiss[a;*], Ariel Felner[b] and Gal A. Kaminka[a]

[a]Bar-Ilan University
[b]Ben-Gurion University

Abstract. The shortest path problem in graphs is a cornerstone of AI theory and applications. Existing algorithms generally ignore edge weight computation time. We present a generalized framework for weighted directed graphs, where edge weight can be computed (estimated) multiple times, at increasing accuracy and run-time expense. This raises several generalized variants of the shortest path problem. We introduce the problem of finding a path with the tightest lower-bound on the optimal cost. We then present two complete algorithms for the generalized problem, and empirically demonstrate their efficacy.

1 Introduction

The canonical problem of finding the shortest path in a directed, weighted graph is fundamental to artificial intelligence and its applications. The *cost* of a path in a weighted graph, 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, constrained optimization, and more.

A common assumption made by existing search algorithms is that the edge weights are determined in negligible (or very small constant) time. However, recent advances challenge this assumption. This occurs when weights are determined by queries to remote sources, or when the graph is massive, and is stored in external memory (e.g., disk). In such cases, additional data-structures and algorithmic modifications are needed to optimize the order in which edges are visited, i.e., optimizing access to external memory [21, 8, 9, 13, 11, 12, 20]. Similarly, when edge weights are computed dynamically using learned models, or external procedures, it is beneficial to delay weight evaluation until necessary [1, 16, 15, 14].

A concrete example serves to illustrate the setting we address. Consider searching for the fastest route between two cities, where edges represent roads, and edge weights represent current travel times, which are queried from an online source (e.g., Google maps). Even a few milliseconds for each query makes the weight evaluation a significant component in the search run-time. Travel times can be estimated even more accurately with information from additional sources (e.g., weather conditions, road curvature and elevation), but their use may significantly increase edge weight computation time.

We present a novel approach to handling expensive weight computation by allowing the search algorithms to incrementally use multiple *weight estimators*, that compute the edge weight with increasing accuracy, but also at increasing computation time. Specifically, we replace edge weights with an ordered set of estimators, each providing a lower and upper bound on the true weight. A search algorithm may quickly compute loose bounds on the edge weight, and invest more computation on a tighter estimator later in the process. In the example above, a local database can be queried quickly to get rough bounds on the travel times (based on the fixed distance and speed limits). Incrementally, online queries and computations can be used as needed to get more accurate edge weight estimations, at increasing computational expense. This approach follows the recently suggested concept of dynamic estimation during planning [22, 23] and its first implementation in AI planning [25].

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 equal. 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.

In this paper, we introduce the *shortest path tightest lower-bound* (SLB) problem, 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 multiply-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) [3]. 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. Experiments demonstrate the dramatic computational savings they offer compared to the baseline which uses the most accurate and expensive estimates in all cases.

2 Background and Related Work

To put this research in context, consider the *abstract* components of the search run-time T, in a manner inspired by [14]:

$$T = T_w + T_v = \tau_w \times w + \tau_v \times v, \tag{1}$$

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

* Corresponding Author. Email: eyal.weiss@biu.ac.il

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 [1, 16, 15, 14], even at the cost of increasing v. Related challenges arise in planning, where action costs can be computed by external (lengthy) procedures [2, 6, 4], or when multiple heuristics have different run-times [10].

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) [21, 8, 9, 13, 11, 12, 20]. This reduces τ_w by amortizing the computation costs, but still assumes a single weight computation per edge.

The approach we take in this paper 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

$$T_w = \tau_{w_1} \times w_1 + \tau_{w_2} \times w_2 \dots \tau_{w_k} \times w_k, \quad (2)$$

with $\tau_{w_1} < \dots < \tau_{w_k}$, and possibly, $w_1 + \dots + w_k > w$. However, the total sum in Eq. 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 run-time 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 [5]. Fuzzy weights [17] allow quantification of uncertainty by grouping approximate weight ranges to several representative sets. Multi-objective weights [19] 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.

3 Shortest Path with Estimated Weights

A standard *weighted digraph* is a tuple (V, E, c), where V is a set of vertices, 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 vertices in V. A *path* $p = \langle e_1, \dots, e_n \rangle$ 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 p is then defined to be $c(p) := \sum_{k=1}^{n} c(e_k)$. The *Goal-Directed Single-Source Shortest Path* (GDS^3P) problem involves finding a path π from a start vertex to a goal vertex, with minimal $c(\pi)$, denoted as C^*.

We now replace the cost function c by an estimator-generating function Θ, which for every edge e yields a sequence of estimation procedures, each providing a lower and upper bound on the weight of the edge (Def. 1). The procedures are ordered by *increasing running times* and assumed to *yield increasingly tightening bounds*.

Definition 1 *A* **cost estimators function** *for a set of edges E, denoted as Θ, maps every edge $e \in E$ to a finite and non-empty sequence of* weight estimation procedures,

$$\Theta(e) := (\theta_e^1, \dots, \theta_e^{k(e)}), k(e) \in \mathbb{N}, \quad (3)$$

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 $i < j$.*

This allows us to define *estimated weighted digraphs*:

Definition 2 *An* **estimated weighted digraph** *is a tuple $G = (V, E, \Theta)$, where V, E are sets of vertices and edges, resp., and Θ is a* **cost estimators function** *for E.*

A path $p = \langle e_1, ..., e_n \rangle$ can now be characterized by the accumulated lower- or upper- bounds on the edges, resulting from the application of *some* weight estimators (Def. 3):

Definition 3 *Let $\Phi(e)$ be a non-empty subset of estimators from the sequence $\Theta(e)$, for an edge e. We denote the* tightest *bounds on $c(e)$, over all estimators in $\Phi(e)$, as $l_{\Phi(e)}$ (maximum lower bound) and $u_{\Phi(e)}$ (minimum upper bound):*

$$\begin{aligned} l_{\Phi(e)} &:= \max\{l_e^i | \theta_e^i = (l_e^i, u_e^i) \in \Phi(e)\} \\ u_{\Phi(e)} &:= \min\{u_e^j | \theta_e^j = (l_e^j, u_e^j) \in \Phi(e)\} \end{aligned} \quad (4)$$

For a path p, let $\Phi(p) := \bigcup_{e \in p} \Phi(e)$. The path lower bound *and* path upper bound *of p w.r.t. $\Phi(p)$ follow, respectively, from the tightest edge bounds defined above.*

$$l_{\Phi(p)} := \sum_{i=1}^{n} l_{\Phi(e_i)}, \quad u_{\Phi(p)} := \sum_{i=1}^{n} u_{\Phi(e_i)} \quad (5)$$

We denote by $\Phi^*(p)$ the maximal size $\Phi(p)$*, which includes* **all** *estimators for edges in p.*

Estimated weighted digraphs and their path bounds generalize the familiar weighted digraphs, which are a special case 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 equal to the weight $c(e)$. In this special case, a shortest tightly-bounded path π in the graph is a solution to a GDS^3P problem. However, in the general case, multiple estimators exist, and we are not guaranteed that every weight can be estimated precisely, even if all estimators for it are used. Thus, several variants of the shortest path problem exist, which correspond to the tightest bounds for the shortest path.

We focus on the *shortest path tighest lower bound* (SLB) problem (Prob. 1). This problem deals with determining a path that achieves L^*—the optimal tightest lower bound on the cost of the shortest path.

Problem 1 (SLB, finding L^*) *Let $P = (G, v_s, V_g)$, where G is an estimated weighted digraph with cost estimators functions Θ, $v_s \in V$ is the start (source) vertex and $V_g \subset V$ is a set of goal vertices. The* **Shortest path tightest Lower Bound** *problem (SLB) is to find a path π from v_s to any goal vertex $v \in V_g$, such that π has the lowest tightest lower bound of any path from v_s to $v \in V_g$, w.r.t. Θ, i.e., $l(\pi) = L^*$ with*

$$L^* := \min_{\pi'} \{l_{\Phi^*(\pi')} \mid \pi' \text{ is a path from } v_s \text{ to } v \in V_g\}. \quad (6)$$

The use of the $\min$ operator may seem counter-intuitive, as typically the tightest lower bound would be the maximal of all lower bounds. Indeed, ideally, we should use $l_{\Phi^*(\pi^*)}$, the tightest (maximal) lower bound of the shortest path π^*. However, π^* is unknown (as the cost function itself is unknown). Thus, instead we have to use L^*, the *minimal* tightest lowest bound of *any* path that leads from v_s to a goal vertex. Necessarily, the use of L^* bounds $l_{\Phi^*(\pi^*)}$ from below, and on the other hand it is the best (maximal) lower bound we may use, when the true edge costs are unknown.

The SLB problem (Problem 1) is a generalization of the standard shortest path problem GDS^3P (Thm. 1), and thus its complexity is at least that of GDS^3P.

Theorem 1 *Problem 1 generalizes GDS^3P problems.*

Proof. We show that *any* standard GDS^3P problem can be formulated as a special case of SLB. In this special case, each edge has one estimator (namely, $k(e) = 1$ for every e), that returns the exact cost (i.e., $l_e^1 = c(e) = u_e^1$), as this implies $L^* = C^*$. A solution for an SLB instance as described above has lower bound L^* and will therefore have cost C^*, hence by definition it is a shortest path. □

The solution to an SLB problem is important as generally exact edge costs may not be known and then L^* provides a tight lower bound for the cost of *any* solution, thus quantifying cost uncertainty. Yet it has additional uses. For example, L^* is key in determining optimal and bounded-suboptimal shortest path solutions. To see this, we recall the definition of admissible solutions: a solution π to 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.,

$$c(\pi) \leq C^* \times \mathcal{B}. \tag{7}$$

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

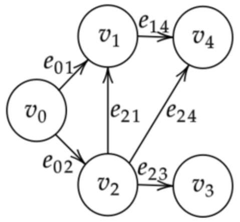
In estimated weighted digraphs the cost $c(\pi)$ of a path π is not known precisely (in the general case), and thus Inequality 7 cannot be shown directly. Instead, as $c(\pi) \leq u_{\Phi^*(\pi)}$ holds, we may prove that π is $\mathcal{B}$-admissible by showing that $u_{\Phi^*(\pi)} \leq C^* \times \mathcal{B}$. Still, the optimal cost C^* is also unknown, so we instead compare to L^* (the SLB solution). Necessarily, $L^* \leq C^*$ holds, thus showing

$$u_{\Phi^*(\pi)} \leq L^* \times \mathcal{B} \tag{8}$$

is sufficient to prove that π is $\mathcal{B}$-admissible (see Example 1). Solving SLB is therefore critical to identifying $\mathcal{B}$-admissible paths with estimated costs (which, for $\mathcal{B} = 1$ are shortest paths).

Example 1 *Consider an estimated weighted digraph $G = (V, E, \Theta)$, with $V = \{v_0, v_1, v_2, v_3, v_4\}$, and $E = \{e_{01}, e_{02}, e_{14}, e_{21}, e_{23}, e_{24}\}$ (see Fig. 1). Here, Θ is defined by the following estimators: For edge e_{01}: $\theta^1_{e_{01}} = (4, 4)$. For edge e_{02}, $\theta^1_{e_{02}} = (2, 6)$, and $\theta^2_{e_{02}} = (3, 5)$. For edge e_{14}, $\theta^1_{e_{14}} = (1, 10)$, $\theta^2_{e_{14}} = (4, 6)$. For edge e_{21}, $\theta^1_{e_{21}} = (2, 3)$, $\theta^2_{e_{21}} = (3, 3)$. For edge e_{23}, $\theta^1_{e_{23}} = (5, 9)$, $\theta^2_{e_{23}} = (7, 8)$. Finally, for edge e_{24}, $\theta^1_{e_{24}} = (4, 6)$. Additionally, the true edge costs have the following values: $c_{01} = 4, c_{02} = 4, c_{14} = 5, c_{21} = 3, c_{23} = 7$ and $c_{24} = 6$.*

Given the graph above, we may define the SLB 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 $\pi^* = \langle e_{01}, e_{14}\rangle$, and the tightest lower bound for C^* is $L^* = l_{\Phi^*(\pi_1)} = l^2_{02} + l^1_{24} = 7$ with $\pi_1 = \langle e_{02}, e_{24}\rangle$ (the SLB solution). Then, considering the solution $\pi_2 = \langle e_{02}, e_{23}\rangle$, one can obtain its tightest upper bound $u_{\Phi^*(\pi_2)} = u^2_{02} + u^2_{23} = 13$ and use it to get its admissibility factor $\mathcal{B}(\pi_2) = u_{\Phi^*(\pi_2)}/L^* = 13/7$.



	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)	(5, 9)	(4, 6)
θ^2		(3, 5)	(4, 6)	(3, 3)	(7, 8)	

Figure 1. Left: Digraph of Example 1. Right: costs and estimates.

4 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.

4.1 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 parameters later.

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^i_e 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

Algorithm 1 BEAUTY

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

1: $g_l(s_0) \leftarrow 0$; OPEN $\leftarrow \emptyset$; CLOSED $\leftarrow \emptyset$
2: Insert s_0 into OPEN with $g_l(s_0)$
3: **while** OPEN $\neq \emptyset$ **do**
4:   $n \leftarrow$ pop node n from OPEN with minimal $g_l(n)$
5:   **if** $Goal(n)$ **then**
6:     $l(\pi) \leftarrow g_l(n)$
7:     $Opt, \underline{l}^*, \bar{l}^* \leftarrow$ BEAUTY-PS
8:     **return** $trace(n), Opt, \underline{l}^*, \bar{l}^*$
9:   Insert n into CLOSED
10:   **for each** successor s of n **do**
11:     **if** s not in OPEN $\cup$ CLOSED **then**
12:       $g_l(s) \leftarrow \infty$
13:     $\tilde{g}_l \leftarrow g_l(n)$
14:     **while** $\tilde{g}_l < g_l(s)$ **and** estimators remain for $e = (n, s)$ **do**
15:       $l(e) \leftarrow$ Apply next estimator for e
16:       $\tilde{g}_l \leftarrow g_l(n) + l(e)$
17:       **if** $\tilde{g}_l > l_{est}$ **then**
18:         break // exit while loop
19:     **if** $\tilde{g}_l < g_l(s)$ **and** $\tilde{g}_l \leq l_{prune}$ **then**
20:       $g_l(s) \leftarrow \tilde{g}_l$
21:       **if** s in OPEN **then**
22:         Remove s from OPEN
23:       Insert s into OPEN with $g_l(s)$
24: **return** $\emptyset, false, \infty, \infty$

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* [18]). 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 1), *sound* (Lemma 2), and *optimal* (Lemma 3).

Procedure 2 BEAUTY-PS

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

1: $Opt \leftarrow true$; $\underline{l}^* \leftarrow l(\pi)$
2: **for each** edge e in π **do**
3:   **if** estimators remain for e **then**
4:     $l \leftarrow$ Apply the best estimator for e
5:     $l(\pi) \leftarrow l(\pi) + l - l(e)$
6:     $l(e) \leftarrow l$
7: **if** $l(\pi) > \underline{l}^*$ **then**
8:   $Opt \leftarrow false$
9: $\bar{l}^* \leftarrow l(\pi)$
10: **return** Opt, $\underline{l}^*$, $\bar{l}^*$

Lemma 1 (Conditional Completeness Prob. 1) 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 2 (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 1 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 3 (Conditional Optimality Prob. 1) 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 2, 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 1–3 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 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.

4.2 *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 2), 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 2). 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 run-time (due to less insertion operations, and cheaper insert/delete operations).

Algorithm 3 A-BEAUTY

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

$\underline{l}^* \leftarrow 0$; $\bar{l}^* \leftarrow \infty$; $Opt \leftarrow false$
while not Opt **do**
  $\pi, Opt, \underline{l}^*, \bar{l} \leftarrow$ BEAUTY $(P, \underline{l}^*, \bar{l}^*)$
  if $\pi = \emptyset$ **then**
    return $\emptyset, \infty$
  if $\bar{l} < \bar{l}^*$ **then**
    $\bar{l}^* \leftarrow \bar{l}$
  Print $\pi, \underline{l}^*, \bar{l}^*$
return $\pi, \bar{l}^*$

Theorem 2 (Completeness, Soundness and Optimality Prob. 1) 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 1 and 2 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 3), which necessarily occurs after a finite number of iterations. □

The proof of Thm. 2 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 $\langle e_{02}, e_{24}\rangle, 7$.*

5 Empirical Evaluation

The theoretical guarantees of BEAUTY and A-BEAUTY touch on their optimality and completeness, but do not provide information as to the run-time savings they offer. We therefore empirically evaluate

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

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 $l_e^1 := c_{old} \times f_1, l_e^2 := c_{old} \times f_2, l_e^3 := c_{old} \times f_3$ 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 taken according to the result of a simple hash function, that depends on $c_{old}(e)$ and a user-input seed, described as follows:

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

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 [24].

We note that the configurations depicted in Table 1 that are chosen according to the hash function of Eq. 9 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 [24]. a C++ planner that extends Fast Downward (FD) [7] (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.

5.1 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.

5.2 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

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

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**

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 run-time 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. 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 run-time $T_2 = 0.5 \times T_w + 2 \times T_v = 7 \times T_v$ vs. $T_1 = T_w + T_v = 11 \times T_v$, i.e., a reduction of $\approx 36\%$ in run-time.

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 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.

5.3 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.

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

5.4 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.

6 Conclusions

This paper presents a generalized framework for *estimated weighted directed graphs*, where the cost of each edge can be estimated by multiple estimators, where every estimator has its own run-time and returns lower and upper bounds on the edge weight. This allows to address novel settings of combinatorial search problems that support an explicit trade-off of search and estimation time. We focus on the *shortest path tightest lower bound* (SLB) problem, which we formally define. SLB problems involve finding a path with the tightest lower bound on the optimal cost. We present two algorithms for solving SLB problems in a guaranteed manner. Experiments reveal the dramatic computational savings they offer.

There are many directions for future research. We believe the performance of the algorithms can be further improved (e.g., by utilizing priors on estimation times to choose estimators across edges). We plan to investigate shortest path variants that minimize path upper-bound and $\mathcal{B}$-admissibility. Extensions for undirected graphs and for informed search are also of significant interest.

Acknowledgements

The research was partially funded by ISF Grant #2306/18 and BSF-NSF grant 2017764. Thanks to K. Ushi. This research was also supported by ISF grant #909/23 to Ariel Felner. Eyal Weiss is supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities and by Bar-Ilan University's President Scholarship.

References

[1] 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, pp. 459–467, (2016).

[2] 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*, 99–115, (2012).

[3] 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, pp. 47–51, (2011).

[4] 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*, pp. 4294–4301, (2017).

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

[6] 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, pp. 65–73, (2012).

[7] Malte Helmert, 'The Fast Downward planning system', *Journal of Artificial Intelligence Research*, **26**, 191–246, (2006).

[8] 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).

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

[10] 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).

[11] R.E. Korf, 'Minimizing disk I/O in two-bit breadth-first search', in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 23, pp. 317–324, (2008).

[12] 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*, pp. 610–616, (2016).

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

[14] 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*, pp. 745–753, (2019).

[15] 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, pp. 476–484, (2018).

[16] 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, pp. 522–530, (2017).

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

[18] 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).

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

[20] Nathan R Sturtevant and Jingwei Chen, 'External memory bidirectional search.', in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-16*, pp. 676–682, (2016).

[21] Jeffrey Scott Vitter, 'External memory algorithms and data structures: dealing with massive data', *ACM Computing Surveys*, **33**(2), 209–271, (June 2001).

[22] Eyal Weiss, 'A generalization of automated planning using dynamically estimated action models–dissertation abstract', in *32nd International Conference on Automated Planning and Scheduling*, pp. 1–3, (2022).

[23] 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).

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

[25] Eyal Weiss and Gal A. Kaminka, 'Planning with multiple action-cost estimates', in *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, pp. 427–437, (2023).