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

Eyal Weiss[1], **Ariel Felner**[2], **Gal A. Kaminka**[1]

[1]The MAVERICK Group, Bar-Ilan University, Israel
[2]Ben-Gurion University, Israel
eyal.weiss@biu.ac.il, felner@bgu.ac.il , galk@cs.biu.ac.il

Abstract

The shortest path problem in graphs is a cornerstone of AI theory and applications. Existing algorithms generally ignore edge weight computation time. In this paper 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 a generalized shortest path problem that optimize different aspects of path cost and its uncertainty. We present a complete anytime solution algorithm for the generalized problem, and empirically demonstrate its 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, constraint satisfaction and optimization, and more.

A common assumption made by existing search algorithms is that the edge weights are determined in negligible 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., the access patterns (Vitter 2001; Hutchinson, Maheshwari, and Zeh 2003; Jabbar 2008; Korf 2008a,b, 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, Salzman, and Srinivasa 2018; Mandalika et al. 2019).

For example, 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 taking a few milliseconds for each query makes the weight evaluation a significant component in the search run-time. The estimated travel times can be more accurately computed with more information: current weather conditions, road curvature and elevation, etc., but the use of these further increases the edge weight computation time.

The example provided above is representative of the current trend of using data-driven models in planning, which introduces various forms of uncertainty to action models, and alternate modeling options—that may in particular provide different estimates for action costs. Reliably quantifying the uncertainty of obtained plans in a scalable manner is thus a matter of high importance, which motivates our work.

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 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.

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), search algorithms should address finding paths whose bounds on the shortest-path cost are optimal in some aspect.

In particular, we wish to determine whether the cost of a given path between two vertices is optimal, or within some suboptimality bound. We show this requires solving the *shortest path tightest lower-bound* (SLB) problem, which involves finding a path with the tightest lower bound on the optimal cost. We present BEAUTY, an uninformed search algorithm based on uniform-cost search (UCS, a variant of *Dijksra's* algorithm). We then use it to construct an iterative complete anytime algorithm (A-BEAUTY) which is guaranteed to solve SLB problems. The algorithms and theoretical guarantees are discussed in detail. Experiments demonstrate the dramatic computational savings they offer with respect to the baseline which computes the true weight in all cases.

2 Background and Related Work

Weighted graphs are used in numerous computational problems. Over the years, the definition of weights have been extended in multiple ways. 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-dimensional weights (Loui 1983) allow each edge to be associated with a vector of different weights, facilitating optimization of multiple objectives. All of these ignore the weight *computation time*, in contrast to our work.

Inspired by (Mandalika et al. 2019), we consider the *abstract* components of the run-time T, of search algorithms:

$$T = \tau_w \times n + \tau_s \times m, \tag{1}$$

where n, m are the number of edge weight computations conducted and number of vertices encountered during the search, resp., and τ_w, τ_s are the average *edge weight computation time* and average *vertex search operations* time (i.e., operations that take place for every vertex considered, such as expansion and priority queue operations).

We can look at different algorithmic approaches in terms of their efforts to reduce n or m, often 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 only on reducing m. 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 calling time-consuming processes, validating geometric and kinematic constraints. Thus these algorithms reduce n (by explicitly delaying weight computations), even at the cost of increasing m (Dellin and Srinivasa 2016; Narayanan and Likhachev 2017; Mandalika, Salzman, and Srinivasa 2018; Mandalika et al. 2019). Related challenges arise in planning, where action costs can be computed by external (lengthy) procedures (Dornhege et al. 2009; Gregory et al. 2012; Francès et al. 2017), or when multiple heuristics have different run-times (Karpas et al. 2018).

There are also approaches that seek to change τ_w (rather than to reduce n), though not explicitly as we do in this paper. τ_w can be high when the graph is too large to fit in random-access memory, and is stored in external memory (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, Maheshwari, and Zeh 2003; Jabbar 2008; Korf 2008a,b, 2016; Sturtevant and Chen 2016). This reduces τ_w by amortizing the computation costs, but still assumes a single weight per edge.

The approach we take in this paper is complementary to those above, and it follows the recent works in dynamic estimation during planning (Weiss and Kaminka 2023b; Weiss 2022; Weiss and Kaminka 2022). We consider the case where the weight of each edge can be estimated multiple times, successively more accurately (and at greater expense). The component $\tau_w \times n$ is then replaced with $(\tau_{w_1} \times n_1 + \tau_{w_2} \times n_2 \ldots \tau_{w_k} \times n_k)$, with $\tau_{w_1} < \ldots < \tau_{w_k}$. Search algorithms—such as those presented in this paper—can make use of this to balance search effort and edge evaluation in a novel, more refined manner, and thus to reduce the overall run-time.

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* $p = \langle e_1, \ldots, e_n \rangle$ 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 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 is the problem of finding a path π from a start vertex to a goal vertex, with minimal $c(\pi)$.

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*, which we expect, w.l.o.g, 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*,*

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

where **estimator** *θ_e^i, if applied, returns lower- and upper-bounds (l_e^i, u_e^i) on $c(e)$, such that $0 \le l_e^i \le c(e) \le u_e^i < \infty$). $\Theta(e)$ is ordered by the increasing running time of θ_e^i.*

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 a set 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):*

$$\begin{aligned} l_{\Phi(e)} &:= \max\{l_e^i | \theta_e^i = (l_e^i, u_e^i) \in \Phi(e)\} \\ u_{\Phi(e)} &:= \min\{u_e^j | \theta_e^j = (l_e^j, u_e^j) \in \Phi(e)\} \end{aligned} \tag{3}$$

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.*

$$l_{\Phi(p)} := \sum_{i=1}^{n} l_{\Phi(e_i)}, \quad u_{\Phi(p)} := \sum_{i=1}^{n} u_{\Phi(e_i)} \tag{4}$$

We denote by $\Phi^*(p)$ the maximal $\Phi(p)$*, 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 $\theta_e = (c(e), c(e))$ with lower and upper bounds equal to the weight $c(e)$. The tightest bounds for the cost of a path π then converge to the standard path cost $c(\pi)$. In this special case, we may then state that π is a **$\mathcal{B}$-admissible shortest path** if $c(\pi)$ is bounded by a suboptimality factor $\mathcal{B}$, i.e.,

$$c(\pi) \leq c^* \times \mathcal{B} \tag{5}$$

where c^* is the cost of the shortest path, a solution to a GDS^3P problem. If $\mathcal{B} = 1$, then π is a shortest path.

However, in the general case, the cost $c(\pi)$ of a path π may not be known precisely, and thus Inequality 5 cannot be shown directly. Instead, as $c(\pi) \leq u_{\Phi^*(\pi)}$, 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 tightest lower bound on the cost of the shortest path (see below). Necessarily, $l^* \leq c^*$, thus showing

$$u_{\Phi^*(\pi)} \leq l^* \times \mathcal{B} \tag{6}$$

is sufficient to prove that π is $\mathcal{B}$-admissible.

In other words, the key step in identifying $\mathcal{B}$-admissible paths with estimated costs (which, for $\mathcal{B} = 1$ are shortest paths) is finding the tightest lower bound on the cost of the shortest path, l^*. To do this, we re-define the familiar GDS^3P problem, so that we search for the shortest-path tightest lower bound.

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*

$$l^* := \min_{\pi'} \{ l_{\Phi^*(\pi')} \mid \pi' \text{ is a path from } v_s \text{ to } v \in V_g \}. \tag{7}$$

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. 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, so it is valid for testing $\mathcal{B}$-admissibility, and on the other hand it is the best (maximal) lower bound we may use, when the true edge costs are unknown.

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 $c_{24} = 6$.*

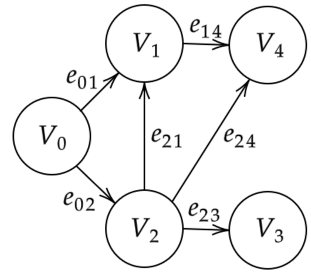

Figure 1: The digraph of Example 1.

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 $\pi^* = \langle e_{01}, e_{14} \rangle$; the tightest lower bound attainable for c^* is $l^* = l_{\Phi^*(\pi)} = l^2_{02} + l^1_{24} = 7$ with $\pi = \langle e_{02}, e_{24} \rangle$; the tightest upper bound for the cost of π^* is $u_{\Phi^*(\pi^*)} = u^1_{01} + u^2_{14} = 10$ and thus the best attainable suboptimality factor $\mathcal{B}$ for the path π^* is $\mathcal{B} = u_{\Phi^*(\pi^*)}/l^* = 10/7$.

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 (Generality)**.** *Problem 1 generalizes the GDS^3P problem.*

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., $l^1_e = c(e) = u^1_e$), as this implies $l^* = c^*$. Solutions corresponding to them achieve the minimum l^* will therefore have cost c^*, hence by definition are shortest paths. $\square$

Given a solution to an SLB problem, and a target $\mathcal{B}$ admissibility factor, we can identify $\mathcal{B}$-admissible solutions, including for the case $\mathcal{B} = 1$, the shortest path. The next section discusses algorithms for solving the SLB problem.

4 Algorithms for Shortest Path Lower Bound

We present two algorithms for solving SLB problems, while reducing the number of estimators used, compared to a standard uniform-cost search algorithm (UCS), which ignores the run-time of estimators. 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.

Algorithm 1: BEAUTY

Input: Problem $P = (G, \Theta, v_s, V_g)$
Parameter: Procedure Get-θ, 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:     $n \leftarrow$ Remove top node from OPEN
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:         $\theta \leftarrow$ Get-θ $(e = (n, s))$
15:         **while** $\tilde{g}_l < g_l(s)$ **and** $\theta \neq \emptyset$ **do**
16:             $l \leftarrow$ Apply(θ)
17:             Cache l for e
18:             $\tilde{g}_l \leftarrow g_l(n) + l$
19:             **if** $\tilde{g}_l > l_{est}$ **then**
20:                 Break
21:             $\theta \leftarrow$ Get-θ (e)
22:         **if** $\tilde{g}_l < g_l(s)$ **and** $\tilde{g}_l \leq l_{prune}$ **then**
23:             $g_l(s) \leftarrow \tilde{g}_l$
24:             **if** s in OPEN **then**
25:                 Remove s from OPEN
26:             Insert s into OPEN with $g_l(s)$
27: **return** $\emptyset, false, \infty, \infty$

Algorithm 1. BEAUTY receives an SLB problem, a procedure Get-θ that maps an edge e to an unused estimator from $\Theta(e)$, and the hyper-parameters l_{est}, l_{prune}. It works in two stages: First, it utilizes as many estimators as needed whenever it encounters a new edge, in order to determine a best path up to value l_{est}. Then, it continues the search with minimal estimations until a solution is found, pruning any path with accumulated lower bound cost greater than l_{prune}.

When a solution π is found by the goal-checking $Goal$ 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 $\bar{l}^* = \underline{l}^*$, namely the path bounds were already tight before BEAUTY-PS, then it determines that π is optimal and sets $Opt \leftarrow true$. 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 $trace$).

Except for the usage of l_{est}, l_{prune} and BEAUTY-PS, BEAUTY is structurally similar to UCS. The data structures OPEN and CLOSED are priority queues, and g_l is a mapping analogous to g in UCS. The primary modification is the addition of an estimation loop that takes place in lines 11–21 (including initialization).

Procedure 2: BEAUTY-PS

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

$Opt \leftarrow true$; $\underline{l}^* \leftarrow l(\pi)$
for each edge e in π **do**
    $\theta \leftarrow$ Get-θ (e)
    while $\theta \neq \emptyset$ **do**
        $l \leftarrow$ Apply(θ)
        Update $l(\pi)$ using l, e and cache
        Cache l for e
        $\theta \leftarrow$ Get-θ (e)
if $l(\pi) > \underline{l}^*$ **then**
    $Opt \leftarrow false$
$\bar{l}^* \leftarrow l(\pi)$
return Opt, $\underline{l}^*$, $\bar{l}^*$

Depending on the hyper-parameters l_{prune}, l_{est}, BEAUTY is *complete* (Lemma 1), *sound* (Lemma 2), and *optimal* (Lemma 3).

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. accumulated lower bound for 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 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 accumulated 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 specific 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 guaranteed to be 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 accumulated 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 $\underline{l}^* \geq l_{est} + \epsilon$ 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 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 ∞. 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.

Example 2. *Consider calling* BEAUTY *with $l_{est} = l_{prune} = \infty$ 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 $\langle e_{02}, e_{24}\rangle, true, 7, 7$.*

Algorithm 3: A-BEAUTY

Input: Problem $P = (G, \Theta, v_s, V_g)$
Parameter: Procedure Get-θ
Output: Path π, bound 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, Get-θ, $\underline{l}^*, \bar{l}^*$)
  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 $\pi, \bar{l}^*$

Algorithm 3. A-BEAUTY (Anytime BEAUTY) iteratively calls 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 $\underline{l}^* > l_{est}$ (due to 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, due to Lemma 2), so that by using the lowest value of $\bar{l}^*$ obtained, 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). Tightened l_{prune} values decrease the size of OPEN, thus reducing memory consumption and run-time (due to less insert operations, and cheaper insert/delete operations). As another optimization, estimations are cached, so that it is not necessary to re-apply estimators. Technically, this is obtained by defining Get-θ to first look for cached values and only then turn to unused estimators. Overall, caching moderates the increase in run-time consumption (due to utilizing more estimations) between subsequent iterations.

Theorem 2 (Completeness, Soundness and Optimality Prob. 1)**.** A-BEAUTY *is complete. If a solution exists for P, then a shortest path 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 for 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 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.

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 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 familiar benchmark set[1], a collection of IPC (International Planning Competition) benchmark instances. 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 specific 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:

$$\text{Hash} = (c_{old}(e) + \text{seed}) \mod 9. \quad (8)$$

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

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

Table 1: The configuration of f_1, f_2, f_3 in rows 2–4 according to the hash values displayed in row 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 2023a).

BEAUTY and A-BEAUTY were implemented as search algorithms in *PlanDEM* (Planning with Dynamically Estimated Action Models (Weiss and Kaminka 2023a), 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 ignores estimator run-time and seeks to maximize weight accuracy, 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 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 number of *expensive estimators* applied, which are the second and third estimators in our experiments (for an edge e these are θ^2_e and θ^3_e). Note that under this setting BEAUTY-PS has nothing to improve, as the solution path is already fully estimated.

We denote by L_2 and L_3 the numbers of second- and third-layer estimators applied during the search. The results are summarized below:

- The ratio $r_{L_2} := L_2(\text{BEAUTY})/L_2(\text{EI-UCS})$ had average of 61.9% (stddev 10.53%), median 61.08%, with overall range spanning 34.75% to 90.65%.

Domain	n_{ins}	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**

Table 2: Summarized performance data of BEAUTY (∞, ∞), A-BEAUTY-2 and A-BEAUTY-10 relative to EI-UCS, with breakdown by unique domains, and where n_{ins} denotes the number of instances per domain. For each algorithm and every domain two entries in the table are presented in the form of average $\pm$ standard deviation in percentage: the ratio of third-layer estimator usage r_{L_3} and the ratio of expanded nodes r_{exp}.

Iteration	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
Final i (%)	0	0	0	0.33	1.09	2.52	10.5	15.65	19.91	50
$\mu(\underline{l}_i^*/l^*)$ (%)	40.01	62.87	76.39	84.8	90.16	93.54	95.41	96.62	97.06	100
$\sigma(\underline{l}_i^*/l^*)$ (%)	7.43	8.78	8.68	7.98	6.8	5.73	4.88	4.07	3.95	0

Table 3: Convergence analysis of A-BEAUTY-10, at iteration number i. Row 2 indicates the number of times convergence occurred in 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.

- 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%.

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, with slightly increased savings for the more expensive third-layer estimators. 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 (the data for r_{L_2} is similar). 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 search effort of EI-UCS (measured by expanded nodes), which is due to the search restart at every iteration. In domains where the estimation savings are similar, it appears that 2 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 computa-

Iteration	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$	$i=10$
μ(pruned/evaluated) (%)	0	0.77	1.8	3.53	10.05	10.71	12.46	14.86	16.68	25.98
σ(pruned/evaluated) (%)	0	4.65	7.31	10.49	15.56	16.17	17.3	18.23	18.77	22.17

Table 4: Pruning analysis of A-BEAUTY-10, at iteration number i. Rows 2 and 3 indicate the mean μ and standard deviation σ, respectively, for the ratio of pruned nodes out of evaluated nodes, in percentages.

tions.

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 occurred in 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 strengthens 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 0.8% 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, 2 iterations of A-BEAUTY offered the same savings as 10 iterations, yet significantly more than a single iteration, it is interesting to examine the role of BEAUTY-PS in improving the results from the first iteration of A-BEAUTY. Recall that Proc. 2 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 as to the effectiveness 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.

6 Discussion

The mathematical framework presented in this paper is quite flexible. Recall that Def. 1 defined a sequence of gradually more accurate and time-consuming estimators. In this paper we utilized the assumption that they must be called in order, but this assumption can easily be lifted, so that e.g., the most accurate and expensive estimator can be invoked first. Generally speaking, this setup makes sense only when there is sufficient information about the running times and expected quality of the estimators, so that more informed choices could be taken. Since this paper introduced a novel framework, we preferred to start with the simpler setting, where the more complex setting is left for future work.

Additional flexibility can be manifested in: estimation times that are fully known, partially known or completely unknown; similarly for expected bounds; and edge costs that can be unknown deterministic constants, or stochastic variables, so long as they are bounded.

7 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. We show that in these settings, deciding whether the cost of a given path is optimal (or within some suboptimality bound) requires solving the *shortest path tightest lower-bound* (SLB) problem, which we 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.

The novel framework offers numerous directions for future research. Certainly, the algorithms presented are first steps, and their performance can probably be improved (e.g., by utilizing knowledge of τ_w to interleave the order of estimator applications across edges). Other algorithmic approaches can be tested as well. Extensions for undirected graphs and for informed search are also of significant interest, as are novel graph search problems that are based on estimated, rather than exact, costs.

Acknowledgements

The research was partially funded by ISF Grant #2306/18 and BSF-NSF grant 2017764. Thanks to K. Ushi. 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

Dellin, C.; and Srinivasa, S. 2016. 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.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009. Semantic attachments for domain-independent planning systems. In *Nineteenth International Conference on Automated Planning and Scheduling*.

Francès, G.; Ramirez, M.; Lipovetzky, N.; and Geffner, H. 2017. Purely Declarative Action Descriptions are Overrated: Classical Planning with Simulators. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*.

Frank, H. 1969. Shortest paths in probabilistic graphs. *operations research*, 17(4): 583–599.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning Modulo Theories: Extending the Planning Paradigm. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hutchinson, D.; Maheshwari, A.; and Zeh, N. 2003. An external memory data structure for shortest path queries. *Discrete Applied Mathematics*, 126(1): 55–82.

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

Karpas, E.; Betzalel, O.; Shimony, S. E.; Tolpin, D.; and Felner, A. 2018. Rational deployment of multiple heuristics in optimal state-space search. *Artificial Intelligence*, 256: 181–210.

Korf, R. 2008a. Linear-time disk-based implicit graph search. *Journal of the ACM*, 55(6): 26–1–26–40.

Korf, R. 2008b. Minimizing disk I/O in two-bit breadth-first search. In *AAAI*, 317–324.

Korf, R. 2016. Comparing Search Algorithms Using Sorting and Hashing on Disk and in Memory. In *IJCAI*, 610–616.

Loui, R. 1983. Optimal paths in graphs with stochastic or multidimensional weights. *Communications of the ACM*, 26(9): 670–676.

Mandalika, A.; Choudhury, S.; Salzman, O.; and Srinivasa, S. 2019. 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*, 745–753.

Mandalika, A.; Salzman, O.; and Srinivasa, S. 2018. 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.

Narayanan, V.; and Likhachev, M. 2017. Heuristic search on graphs with existence priors for expensive-to-evaluate edges. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*.

Okada, S.; and Gen, M. 1994. Fuzzy shortest path problem. *Computers & industrial engineering*, 27(1-4): 465–468.

Sturtevant, N. R.; and Chen, J. 2016. External Memory Bidirectional Search. In *IJCAI-16*.

Vitter, J. S. 2001. External memory algorithms and data structures: dealing with massive data. *ACM Computing Surveys*, 33(2): 209–271.

Weiss, E. 2022. A Generalization of Automated Planning Using Dynamically Estimated Action Models–Dissertation Abstract. In *32nd International Conference on Automated Planning and Scheduling*, 1–3.

Weiss, E.; and Kaminka, G. A. 2022. Position Paper: Online Modeling for Offline Planning. In *Proceedings of the 1st ICAPS Workshop on Reliable Data-Driven Planning and Scheduling*.

Weiss, E.; and Kaminka, G. A. 2023a. PlanDEM. https://github.com/eyal-weiss/plandem-public. Accessed: 2023-03-14.

Weiss, E.; and Kaminka, G. A. 2023b. Planning with Multiple Action-Cost Estimates. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33.