Heuristic Search on Graphs with Existence Priors for Expensive-to-Evaluate Edges

Venkatraman Narayanan, Maxim Likhachev
The Robotics Institute, Carnegie Mellon University
{venkatraman, maxim}@cs.cmu.edu

Abstract

We address the problem of finding shortest paths in graphs where some edges have a prior probability of existence, and their existence can be verified during planning with time-consuming operations. Our work is motivated by real-world robot motion planning, where edge existence is often expensive to verify (typically involves time-consuming collision-checking between the robot and world models), but edge existence probabilities are readily available. The goal then, is to develop an anytime algorithm that can return good solutions quickly by somehow leveraging the existence probabilities, and continue to return better-quality solutions or provide tighter suboptimality bounds with more time. While our motivation is fast and high-quality motion planning for robots, this work presents two fundamental contributions applicable to generic graphs with probabilistic edges. They are: a) an algorithm for efficiently computing all *relevant* shortest paths in a graph with probabilistic edges, and as a by-product the expected shortest path cost, and b) an anytime algorithm for evaluating (verifying existence of) edges in a collection of paths, which is *optimal* in expectation under a chosen distribution of the algorithm interruption time. Finally, we provide a practical approach to integrate a) and b) in the context of robot motion planning and demonstrate significant improvements in success rate and planning time for a 11 degree-of-freedom mobile manipulation planning problem. We also conduct additional evaluations on a 2D grid navigation domain to study our algorithm's behavior.

1 Introduction

Many real world graph search problems involve expensive computation for determining the cost or existence of an edge in the graph. An example is that of robot motion planning, where states in the graph correspond to full configurations of the robot (e.g., a 7 degree-of-freedom manipulator's graph state would be a 7-dimensional vector) and edges between two states exist if the robot can get from the first configuration to the second in a collision-free and kinematically feasible fashion. This existence check requires expensive collision checking between the robot mesh model and the world representation, as well as kinematic feasibility checks, which often turn out to be the most time-consuming parts of the graph search.

In several scenarios however, we do have some prior probabilistic information about the existence of an edge. In the example of robot motion planning, a crude probabilistic model could compute an approximate distance between the robot and the world (without doing the full-blown collision checking) and use that to compute edge existence probabilities. Sophisticated schemes for learning edge existence probabilities online during the graph search are also possible. In this work, we seek to exploit any such probabilistic edge existence information to develop an anytime graph search algorithm that can return feasible solutions quickly. En route to this goal, we develop principled algorithms that are applicable in a variety of related problems. Our contributions are summarized below:

Contribution 1: Shortest Paths in Graphs with Probabilistic Edges
Our problem setup requires finding the shortest path in a graph where some edges have a prior probability of existence as well as a true state (measurable by evaluating those edges). This can be viewed as solving the shortest path problem on a graph which is drawn from a distribution over graphs defined by the edge existence probabilities. Our first algorithm Expected Shortest Paths* (ESP*) efficiently computes the set of all *unique* shortest paths that could result from this distribution, and consequently the expected shortest path cost. To our knowledge, this is the first algorithm that can efficiently (defined more rigorously later) compute the expected shortest path cost for a graph distribution defined by independent Bernoulli edge existence priors.

Contribution 2: Optimal Anytime Edge Evaluation
Assume that we are presented with a set of candidate paths from start to goal that include probabilistic edges. Now at any time, we have the ability to query the true state of an edge, albeit by a time-consuming process. Our second algorithm, Anytime Edge Evaluation (AEE*) answers the question of deciding how to evaluate these edges, so that the suboptimality bound we provide on the solution quality is minimum in expectation whenever the algorithm is interrupted.

Our experiments on an 11 DoF mobile manipulation planning problem for the PR2 robot, as well as evaluations on a grid navigation domain demonstrate the strength of our proposed approach in comparison to existing methods for alleviating the problem of time-consuming collision checking.

2 Related Work

The problem of time-consuming edge evaluation in robot motion planning has been touched upon in a number of recent works. These works adopt one of the following strategies to tackle slow collision-checking: i) lazily build the search graph as in Lazy PRM (Bohlin and Kavraki 2000), ii) use collision probabilities, which could be learnt online during planning (Huh and Lee 2016) to bias sampling-based motion planners, or heuristically guide search-based planners as in POMP (Choudhury, Dellin, and Srinivasa 2016), or iii) use lazy search techniques to defer as many edge evaluations as possible, as in LazySP (Dellin and Srinivasa 2016) or Lazy Weighted A* (Cohen, Phillips, and Likhachev 2014). Our work mostly falls under the last camp, where we use edge existence probabilities and estimated evaluation times to determine how to evaluate edges in a lazy fashion.

Other works related to minimizing edge evaluations include Partial Expansion A* (PEA*) (Yoshizumi, Miura, and Ishida 2000) and its enhancement EPEA* (Felner et al. 2012), and BEAST (Kiesel and Ruml 2016). The former methods minimize unnecessary node insertions (and hence edge evaluations) when dealing with large branching factors, while the latter provides a method for online estimation and utilization of edge-existence probabilities in the context of abstraction-guided sampling-based motion planning. Another well-known problem that bears resemblance to ours is the Canadian Traveler Problem (CTP) (Papadimitriou and Yannakakis 1991). The major difference between the two is that in CTP, the existence status of an edge is known only when the agent physically moves to the location, as opposed to being able to evaluate an edge at any point in our case. Therefore, we deal with a deterministic shortest path problem (albeit with priors for edge existence), as opposed to the CTP—a deterministic Partially Observable Markov Decision Process (POMDP) (Eyerich, Keller, and Helmert 2010).

In comparison to existing lazy search methods, ours provides the ability to compute an optimal edge-evaluation strategy for a given set of candidate paths en masse (which could also be partial paths to the goal), as opposed to just one path. Moreover, our method provides greater flexibility in terms of determining when to evaluate edges and when to continue planning—either after a complete set of candidate paths has been found, after a single candidate path has been found, or using some other arbitrary interleaving strategy.

Finally, our approach has an attractive anytime property in that it is provably optimal under expectation with respect to stochasticity in both edge existences and the interruption time, assuming all interruptions occur only after the candidate set of paths has been found. This is a novel definition of optimal anytime behavior, distinct from the one proposed in (Thayer, Benton, and Helmert 2012), where an ideal anytime algorithm is defined as one that reduces the time between distinct solutions. On the other hand, our notion of ideal anytime behavior in AEE* includes both solution quality and the expected interruption time.

In addition to the above distinguishing features, ESP* is the first algorithm to our knowledge that can efficiently compute the exact expected shortest path cost for a distribution of graphs defined by Bernoulli edge existence probabilities.

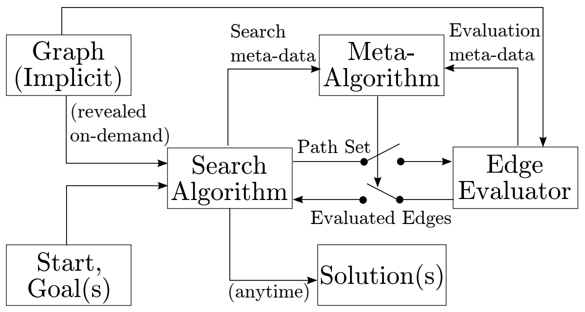

Figure 1: Strategy for interleaving search and edge evaluations. The search module provides a candidate set of paths (not necessarily from start to goal) to the edge evaluator whenever the meta-algorithm commands it to. The evaluated statuses of the edges are then returned to the search algorithm for future use. Also, the search algorithm may choose to publish solutions in an anytime fashion as opposed to just once. Search and evaluation meta-data could include statistics such as time used, number of expansions made, number of paths found, or number of edges evaluated.

3 Overview

The overarching goal of this work is to leverage edge existence probabilities to find solutions as quickly as possible. Right away, this leads to two questions: a) do we care only about the optimal solution or would we rather have an anytime algorithm that returns a feasible path very fast and continues to improve it with time, and b) should the objective be to minimize edge evaluations, or rather minimize the total planning time, in which case we also need to consider the overhead of the search algorithm as well (i.e, operations such as inserting and deleting from a data structure). If we cared only about minimizing edge evaluations, we could find all possible paths between the start and goal, and then use an "optimal" edge evaluator on those paths for some notion of optimality. The other extreme would be to evaluate every edge that is being considered by the search algorithm, which reduces to a typical search implementation.

Our strategy (shown in Fig. 1) is to interleave search and edge evaluations, with flexibility in the choice of how to interleave them. This is similar to the LazySP approach (Dellin and Srinivasa 2016), but has two key novelties : i) the search algorithm can present an arbitrary set of paths to the edge evaluator rather than just 1 path from start to goal, and ii) the search and edge evaluation modules are moderated by a meta-algorithm that determines when each module should be active, instead of using a fixed interleaving policy. The intuition for the first is that the edge evaluator can make more informed decisions by considering paths en masse along with individual existence probabilities of edges. For instance, it might be worthwhile jointly considering the time required for evaluating a path and its probability of existence, if we are interested in an anytime setting. However, a complicated edge evaluator might lead to another time-consuming entity, and that motivates our meta-method. Finally, we assume that the run time of the meta-algorithm is negligible compared to either of the other two modules, so that the planning time is only a function of the search and edge evaluation modules. In the following sections, we will describe our contributions for each module.

4 Expected Shortest Paths* (ESP*)

The first algorithm, ESP*, is towards the search module whose job is to generate a set of (possibly partial) paths for the edge evaluator. Naturally, we would like these paths to be a diverse sampling of shortest, likeliest, and fastest-to-evaluate paths (which could all be different) so that the edge evaluator can hedge its bets. ESP* answers the question: what are *all* the paths we might care about in making the best edge-evaluation decisions under uncertainty? This results in the problem of finding the shortest paths in the graph under all combinations of edge existences—any more information would be unnecessary to the edge evaluator. Incidentally, an algorithm that achieves this could also compute the expected shortest path cost, as discussed next.

4.1 Problem Setup

Formally, we have

- $\Gamma = (X, E)$, a finite graph with vertices X and *possible* edges E. We use x_{start} and x_{goal} to denote the start and goal vertices.
- $c : X \times X \to \mathbb{R}^+$, cost of an edge between two vertices (∞ if an edge does not exist).
- $p : E \to [0, 1]$, Bernoulli prior for existence of $e \in E$.
- $E_b = \{e \in E | p(e) \neq 1\}$, the set of all edges in Γ that are stochastic (i.e, there is some chance that the edge might not exist).
- $k = |E_b|$, the number of stochastic edges in Γ.

The Bernoulli prior p implicitly defines a distribution $P(G)$ over the graphs G that can be generated as sub-graphs of Γ. The problem we consider is that of finding the set of shortest paths across all (2^k) samples drawn from $P(G)$, and consequently the expected shortest path cost c^*_μ for this graph distribution. If we let $c^*(G)$ denote the cost of the shortest path on graph G,

$$c^*_\mu = \mathop{\mathbb{E}}_{G\sim P(G)}[c^*(G)] = \sum_{i=0}^{2^k-1} c^*(G_i)P(G_i), \tag{1}$$

where $P(G_i)$ is the probability that graph G_i is drawn from $P(G)$ (product of the existence and non-existence probabilities of the edges present and absent respectively). Note that c^*_μ is infinite if there is no deterministic path from start to goal.

4.2 Algorithm

While a trivial algorithm to compute c^*_μ would be to find $c^*(G)$ for every realization of G, it is impractical due to the exponential cardinality of the sample space. The insight we leverage here is that the set of *unique* shortest paths S_σ across all the realizations has a much smaller cardinality than 2^k typically, and that it is sufficient to find this set to compute c^*_μ. Clearly, the shortest path amongst all paths in S_σ is the one where all stochastic edges are extant, and the longest path in the set is the one where all stochastic edges are nonexistent. The problem now reduces to finding the in-between paths.

Expected Shortest Paths* (ESP*) presented in Alg. 1 solves this task. At heart, it is very much simply A* search, albeit with two modifications: i) it operates on an augmented graph where every state s in the graph is the original graph

Algorithm 1 ESP*

```
procedure ComputeExpectedCost($S_\sigma$)
    // #Graph instances where all edges traversed by $s_i$ are existent
       (augmented states $s_i$ are paths in the original graph).
    $\nu[1 \dots |S_\sigma|] = 0$
    // Probability of getting $s_i$ as the shortest path.
    $\rho[1 \dots |S_\sigma|] = 0$
    for all $i \in 1 : |S_\sigma|$ do
        $\nu[i] = 2^{k-\text{NumNonZeros}(s_i.b)}$
        $\rho[i] = p(s_i) \cdot \nu[i]$
        for all $j \in 1 : i-1$ do
            if $s_i.b \subset s_j.b$ then
                $\rho[i] = \rho[i] - p(s_j) \cdot \nu[j]$
    return $\sum_{i=1}^{|S_\sigma|} g(s_i) \cdot \rho[i]$
procedure Succ($s$)
    $S = \emptyset$
    for all $e \in$ OutEdges($s.x$) do
        $s' = [e.second, s.b]$
        $p(s') = p(e) \cdot p(s)$
        if $e \in E_b$ and $s'.b[e.index] \neq 1$ then
            $s'.b[e.index] = 1$        ▷ Update $s.b$ if the new edge is stochastic.
        $S = S \cup \{s'\}$
    return $S$
procedure ExpandState($s$)
    Remove $s$ from OPEN
    for all $s' \in$ Succ($s$) do
        $\mathcal{C} = \{z \in \text{CLOSED} | z.x = s'.x \wedge z.b \subseteq s'.b\}$
        if $|\mathcal{C}| \neq 0$ then
            continue   ▷ Prune path if it satisfies dominance condition
        if $s'$ was not seen before then
            $g(s') = \infty$
        if $g(s') > g(s) + \tilde{c}(s, s')$ then
            $g(s') = g(s) + \tilde{c}(s, s')$
            if $s' \notin$ CLOSED then
                Insert/Update $s'$ in OPEN with priority $g(s') + \tilde{h}(s')$
procedure Main()
    $b_{empty} = [0, 0, \dots, 0]$        ▷ Bit-vector representing stochastic edges traversed
    $s_{start} = [x_{start}, b_{empty}]$, $s_{goal} = [x_{goal}, b_{empty}]$
    $p(s_{start}) = 1, p(s_{goal}) = 0$
    $g(s_{start}) = 0,\ g(s_{goal}) = \infty$
    OPEN = $\emptyset$, CLOSED = $\emptyset$
    $S_\sigma = \emptyset$        ▷ Set of paths from $x_{start}$ to $x_{goal}$ ordered by cost
    Insert $s_{start}$ in OPEN with priority $\tilde{h}(s_{start})$
    while $g(s_{goal}) >$ OPEN.MinKey() do
        if OPEN.Empty() then return ($S_\sigma, \infty$)
        $s$ = OPEN.Top()
        CLOSED = CLOSED $\cup \{s\}$
        if $s.x = x_{goal}$ then
            $S_\sigma = S_\sigma \cup \{s\}$
            // Prune paths from OPEN which traverse
            // all stochastic edges traversed by $s$
            OPEN = OPEN $\setminus \{o \in \text{OPEN} | o.b \supseteq s.b\}$
            continue
        ExpandState($s$)
    $S_\sigma = S_\sigma \cup \{s_{goal}\}$
    return ($S_\sigma$, ComputeExpectedCost($S_\sigma$))
```

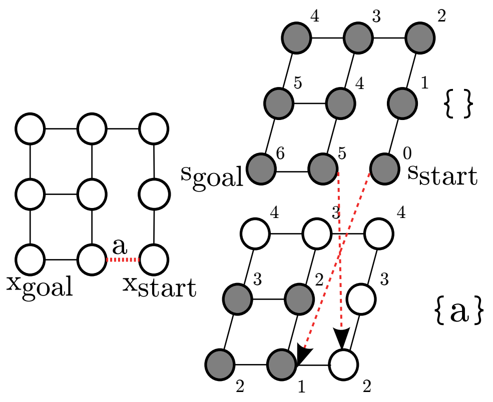

Figure 2: *Left*: An undirected graph with 1 stochastic edge a and unit costs on all edges. *Right*: The augmented graph searched by ESP*. The upper layer corresponds to paths that have not traversed a, and vice versa for the lower one. The numbers next to the vertices represent their g-values and the shaded vertices are those expanded by ESP*, assuming a zero heuristic. Notice how some states in the augmented graph are "pruned" despite having smaller g-values than $g(s_{goal}) = 6$, through the dominance relationships. For e.g., assuming $(0, 0)$ is bottom-left, the state $[(2, 1), \{a\}]$ with a g-value of 3 is dominated by $[(2, 1), \{\})$ with a g-value of 1. Intuitively, this is because the path $(2, 0)$—$(1, 0)$—$(2, 0)$—$(2, 1)$ cannot be part of an optimal solution in any instantiation of the graph (sampled edge existences), given that we know the path $(2, 0)$—$(2, 1)$ is shorter and traverses only a subset (in this case empty set) of stochastic edges in the other path.

node x augmented with a bit vector b of length k that represents which stochastic edges have been traversed by a path to x, and ii) it applies a state dominance relationship to prune partial paths that are guaranteed to not belong in S_σ. Intuitively, the augmented graph (Fig. 2) keeps track of partial paths along with the set of probabilistic edges that are part of those partial paths. This allows us to apply efficient dominance relationships: given two partial paths to an original graph vertex x each of which traverses probabilistic edges a, b and a, b, c (in any order), and have costs 10 and 20 respectively, the latter partial path definitely cannot belong to S_σ. That is, the existence or non-existence of edge c is irrelevant to the shortest path from x_{start} to x_{goal} through state x, for *every* $G \sim P(G)$ in which a and b are existent.

Notation. In the algorithm, $g(s)$ represents the cost-to-come to s from the start state s_{start}, $\tilde{c}(s, s') = c(s.x, s.x')$ is the cost of an edge between s and s', and $\tilde{h}(s) = h(s.x)$ is a consistent heuristic for state $s.x$ in Γ (and hence also consistent for the augmented graph). In addition, $p(s)$ is the probability of arriving at state s from s_{start}, and OPEN is a priority queue sorted by $f(s) = g(s) + \tilde{h}(s)$. The methods OPEN.TOP() and OPEN.MINKEY() return the best state in OPEN and its f-value respectively, while OPEN.EMPTY() returns true if the priority queue is empty. We will use the subset notation $b_1 \subset b_2$ on bit vectors b_1 and b_2 to indicate that the "set" (equal to 1) bits in b_1 are also set in b_2. The method NUMNONZEROS(b) returns the number of set bits in b. Finally, we use an edge struct e with three members: the parent vertex $e.first$, the child vertex $e.second$, and the index of the edge in E_b denoted by $e.index$.

The algorithm proceeds like usual A* except for two main differences: Lines 25–27 codify the dominance pruning through checking if the set of traversed edges in a path to x is a superset of traversed edges in previously computed paths to state x. Lines 47–51 save a newly found path to x_{goal}, and prune any states in OPEN that cannot lead to a better solution given the edges traversed by the newly saved path. Note that the SUCC method not only generates the successor states for an augmented graph state but also computes the arrival probabilities $p(s)$ to those successor states. Finally, the COMPUTEEXPECTEDCOST method processes the set of saved paths and returns the expected shortest path cost.

4.3 Theoretical Analysis

Theorem 1 (Correctness)**.** *The expected shortest path cost returned by the* COMPUTEEXPECTEDCOST *function in ESP* (Line. 54) is equal to* c^*_μ *(Eq. 1).*

Proof. (Sketch) A*'s monotone property (Hart, Nilsson, and Raphael 1968) guarantees that states are expanded in non-decreasing order of f-values, which for states s where $s.x = x_{goal}$, are simply g-values. Thus, if we terminate the search either when OPEN is empty, or when we are about to expand the state s with $s.x = x_{goal}$ and $s.b = [0, 0, \ldots, 0]$ (i.e, corresponding to a deterministic path), we are guaranteed to have found (put in CLOSED) every path to the original goal state x_{goal} that is shorter than the shortest fully deterministic path. The pruning step only removes paths which are longer than an existing path with a subset of traversed stochastic edges, and is therefore admissible. Finally, COMPUTEEXPECTEDCOST simply partitions the 2^k possible graph instances into $|S_\sigma|$ groups with distinct shortest paths for each (in non-decreasing order of path costs) much like a priority encoder, and computes the probabilities of occurrence for each group. □

Theorem 2 (Efficiency)**.** *A vertex x in the underlying graph Γ is expanded at most $2^k \cdot (1 - \frac{1}{2^l}) + 1$ times, where l is the number of stochastic edges in the shortest path from x_{start} to x on Γ.*

Proof. (Sketch) Again, by A*'s monotone property, the first instance a state with underlying original vertex x is expanded will be through a path with l stochastic edges. The pruning step (Line 25) guarantees we will never expand a path to x with a combination of stochastic edges that is a superset of the l stochastic edges. The number of ways we can get to x through a superset of those l edges is 2^{k-l}, implying that the maximum number of additional paths to x that may not get pruned is $2^k - 2^{k-l}$. Including the first path with l stochastic edges, we arrive at $2^k \cdot (1 - \frac{1}{2^l}) + 1$. □

Corollary 1. *If the shortest path from x_{start} to x on Γ is deterministic (i.e, $l = 0$), ESP* expands at most one state corresponding to each vertex x in the underlying graph.*

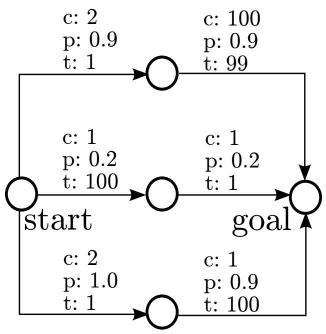

Figure 3: An anytime algorithm must jointly consider the edge costs, probabilities and evaluation times in deciding what edges to evaluate, as well as define explicitly what its desired "anytime" behavior is. In this example, evaluating the shortest (c) or likeliest (p) path will be suboptimal for an anytime algorithm that wants to return a feasible solution as fast as possible. On the other hand, evaluating the path with the smallest expected evaluation time (t) will be suboptimal for an anytime algorithm that tries to minimize its expected solution suboptimality at an arbitrary interruption time.

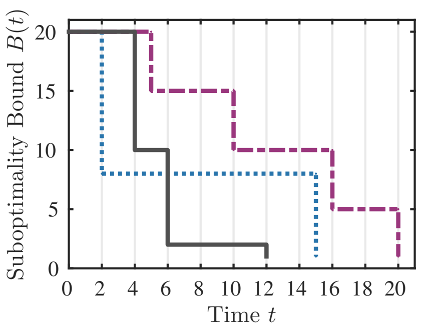

Figure 4: Illustration of AEE*'s optimization objective. Consider three hypothetical anytime profiles (the suboptimality bound as a function of time) corresponding to distinct edge-evaluation policies. AEE* finds the policy corresponding to the solid line, since it has the least area under its curve. Note that each curve is only a possible realization dependent on the edge existence probabilities, and that AEE*'s objective is to minimize the *expected* area under the curve.

Finally, we note that these properties hold only when running optimal A* on the augmented graph. The investigation of these properties when using bounded suboptimal versions such as Weighted A*, as often the case in practice for large graphs, is left for future work.

5 Optimal Policy for Edge Evaluation under Anytime Interruption

Let us now assume we have a set of paths $S_\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ between x_{start} and x_{goal} produced by ESP*, and $c(\sigma_i)$ be the cost of path σ_i, assuming all stochastic edges exist. Let $E = \{e \in \bigcup_{i=1}^{n} \sigma_i | p(e) \neq 1\}$ be the set of all probabilistic edges across the set of paths S_σ and $|E| = k$. Let t_i denote the (estimate of) time taken to evaluate the probabilistic edge $e_i \in E$. Clearly, there are multiple ways (Fig. 3) one can go about evaluating these edges. If all we care about is the optimal path, then we simply order the paths by increasing cost and evaluate edges in each until we find a path that is valid. If instead, we care only about a feasible solution, we would order the paths by likelihood of their existence.

Our next proposed algorithm, Anytime Edge Evaluation* (AEE*) strives to find a balance between conflicting objectives of finding good quality solutions and finding feasible solutions quickly. We formulate the edge-evaluation problem as a Markov decision problem where the objective is to minimize the expected suboptimality bound of the returned solution at any given interruption time. Formally, let a state $s \in \mathcal{S}$ (different from the s used in ESP*) be a ternary k-vector $\{-1, 0, 1\}^k$, where the element $s[i]$ takes the value 1 if edge e_i is valid, -1 if invalid and 0 if unknown (not yet evaluated). In other words, state s captures information about what edges have been evaluated and what the outcomes were. An action $a_i \in \mathcal{A}$ corresponds to evaluating edge e_i. Let $I_\sigma = \{i \mid e_i \in \sigma \wedge p(e_i) \neq 1\}$ denote the index set of stochastic edges in a path σ. For any state s, define $B(s)$ as:

$$B(s) = \frac{\overline{B}(s)}{\underline{B}(s)} \qquad \begin{aligned} \overline{B}(s) &= \min_j c(\sigma_j) \text{ s.t. } \quad s[i] = 1 \; \forall i \in I_{\sigma_j} \\ \underline{B}(s) &= \min_j c(\sigma_j) \text{ s.t. } s[i] \neq -1 \; \forall i \in I_{\sigma_j}, \end{aligned}$$

where we assume the minimum of an empty set is ∞ and $\infty/\infty = 1$. Here, $\overline{B}(s)$ represents the lowest path cost amongst all paths which we know definitely exist (all stochastic edges have been evaluated as valid), and $\underline{B}(s)$ represents the lowest path cost amongst all paths that are not yet proven to be invalid (i.e., those paths may or may not exist based on the current edge evaluations). If the cost of the optimal path is c^*, then we have $\underline{B}(s) \leq c^* \leq \overline{B}(s)$. If the algorithm always returns the path corresponding to the argmin of $\overline{B}(s)$ with cost $c = \overline{B}(s)$ whenever it is interrupted, we have $c \leq B(s) \cdot c^*$, implying that $B(s)$ is a multiplicative suboptimality bound.

From the perspective of an anytime algorithm, we could have the algorithm be interrupted at any time t, which for simplicity is presently assumed to be drawn uniformly at random from $[0, T]$, with T exceeding the time taken to evaluate all stochastic edges. At any such time t, the algorithm can provide a suboptimality bound corresponding to the last published solution before being interrupted. This suboptimality bound is denoted $B(t)$, with some abuse of notation. We now define the decision making problem (i.e, a mapping from state s to action a) as that of choosing actions in a way to minimize the expected suboptimality bound, where the expectation is over the interruption time, as well as the inherent stochasticity in evaluating edges.

Let $p(s'|s, a)$ denote the transition probability, and a policy be denoted by $\pi : \mathcal{S} \to \mathcal{A}$. Then,

$$\pi^* = \underset{\pi}{\text{argmin}} \underset{t \sim U(0,T)}{\mathbb{E}} [B(t)] = \underset{\pi}{\text{argmin}} \frac{1}{T} \int_{t=0}^{T} B(t) dt. \quad (2)$$

Let s_m represent the state at (discrete) step m obtained by following the policy π, $t(s_m)$ represent the (continuous) time at which s_m is reached and $t(a_m)$ represent the time taken to execute action $a_m = \pi(s_m)$. Let us use s_{m+1} to denote the outcome of action a_m.

Now, the bound at time t, $B(t)$ is the suboptimality bound for state s_m, such that $t(s_m) \leq t < t(s_{m+1})$. However note that s_m itself is a random variable drawn from the distribution over states obtained at step m by following the policy π. If $[s_0, s_1, \ldots, s_m, \ldots, s_M]$ denotes a trajectory obtained by following policy π, and $T_\pi(s_m|s_{m-1}) = p(s_m|s_{m-1}, \pi(s_{m-1}))$ the transition probability between states s_{m-1} and s_m under policy π, we have,

$$B(t) = \underset{\substack{s_m\sim \\ T_\pi(s_m|s_{m-1})}}{\mathbb{E}} \left[\sum_{m=0}^{\infty} B(s_m) \cdot \mathbb{1}(t(s_m) \leq t < t(s_{m+1})) \right],$$

where $\mathbb{1}$ is the indicator function which evaluates to 1 when the conditional is true, and 0 otherwise. Using the above in Eq. 2, and dropping the expectation distribution subscript,

$$\begin{aligned}
\pi^* &= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \int_{t=0}^{T} \mathbb{E} \left[\sum_{m=0}^{\infty} B(s_m) \cdot \mathbb{1}(t(s_m) \leq t < t(s_{m+1})) \right] dt \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E} \left[\sum_{m=0}^{\infty} B(s_m) \int_{t=0}^{T} \mathbb{1}(t(s_m) \leq t < t(s_{m+1})) dt \right] \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E} \left[\sum_{m=0}^{\infty} B(s_m) \int_{t=t(s_m)}^{t(s_{m+1})} dt \right] \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E} \left[\sum_{m=0}^{\infty} B(s_m)(t(s_{m+1}) - t(s_m)) \right] \\
&= \underset{\pi}{\operatorname{argmin}} \frac{1}{T} \mathbb{E} \left[\sum_{m=0}^{\infty} B(s_m) t(a_m) \right].
\end{aligned}$$

Defining

$$C(s_m) = \frac{1}{T} B(s_m) t(\pi(s_m)), \tag{3}$$

the optimal policy $\pi^* = \underset{\pi}{\operatorname{argmin}} \mathbb{E} \left[\sum_{m=0}^{\infty} C(s_m) \right]$.

Note that the expectation in this equation is only with respect to transition uncertainty due to stochastic edges (the one due to interruption time has been integrated out). Therefore, the decision making problem has been reduced to a stochastic shortest path (SSP) problem, where we minimize the expected sum of future costs for the specific definition of cost (Eq. 3), and goal states given by $\{s : B(s) = 1\}$[1].

Intuitively, the term $B(s_m)t(\pi(s_m))$ is simply the area of the rectangle with those two corresponding sides, and we seek to find the policy that minimizes the area under the piece-wise constant curve which represents the suboptimality bound as a function of time, with transition-points on the curve corresponding to time-steps at which an edge was evaluated. Figure 4 illustrates this intuition.

In our experiments, we use LAO* (Hansen and Zilberstein 2001) to solve the resulting SSP optimally when it is tractable, or in an online fashion (with fixed time budget for policy computation) when the number of stochastic edges is intractably large. On a practical note, when the set of paths S_σ are only partial paths from x_{start} (not all the way to x_{goal}), we can still use AEE* on these paths by adding the admissible heuristic estimate of the last node on every path to its current cost.

6 Experiments

We evaluate our approach on two domains with distinct properties: the first, a 11 degree-of-freedom mobile manipulation planning problem with dense stochastic edges and the second, a synthetic 2D grid navigation problem with sparse stochasticity. For both domains, we use the augmented graph construction of ESP*, AEE* for edge evaluation, and a meta-algorithm described shortly. An added advantage of using the augmented graph construction is that interleaved search and evaluation can be done without an incremental search algorithm—we only need to update the affected states in OPEN whenever new edge validity information is provided by the evaluator.

Path-Set Selection. While ESP* is capable of producing every relevant path we might care about in theory, it is impractical to wait until complete termination of ESP* because of the exponentially large state space. Consequently, we present two methods to obtain a candidate set of paths at any given time: a) select the set of distinct shortest paths from OPEN corresponding to unique sets of stochastic edges traversed, b) select the set of distinct shortest paths from OPEN according to multiple searches, each with its own heuristic (e.g, Fast Downward Stone Soup (Helmert, Röger, and Karpas 2011) or Multi-Heuristic A* (Narayanan, Aine, and Likhachev 2015)). We use the first strategy for the grid navigation experiments, and the second for mobile manipulation planning. In the latter case, we specifically add an extra search with a heuristic that guides the search along the likeliest path to the goal. Note that in both cases, the set of paths are only partial paths to the goal, but can nevertheless be used in conjunction with AEE* as discussed.

Meta-Algorithm. For both domains, we use a simple meta-algorithm to switch between searching and edge evaluation. If t_{search} and t_{eval} represent the cumulative times used thus far for searching and edge evaluation, we pick the operation with a smaller t at any given instant[2]. This naturally balances time spent searching versus evaluating. More sophisticated methods that try to estimate the time required in the future for each operation might be possible, but as mentioned earlier, the meta-algorithm's run time needs to be negligible compared to other modules.

6.1 Mobile Manipulation Planning

Our first domain is 11 degree-of-freedom (DoF) full-body motion planning for the PR2 robot (a dual-arm mobile manipulation robot). The planner's task is to find a collision free motion for the robot to approach and

[1]It is possible to generalize this result to arbitrary distributions for the interruption time, however with the added complexity that the cost function $C(s)$ is now dependent on the arrival time to s.

[2]The resolution for switching is either one expansion, or one complete run of the AEE* policy on the incumbent set of paths.

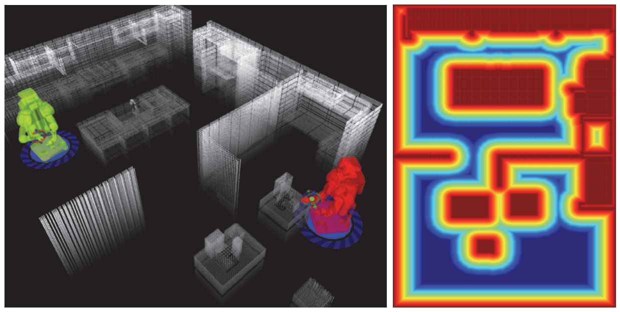

Figure 5: *Left:* The kitchen domain for 11 DoF mobile manipulation experiments with randomly positioned tables and randomized clutter on top of the tables. *Right:* The distance transform $d(x, y)$ on the 2D map which is used to compute edge existence probabilities for mobile manipulation.

pick up objects on cluttered tables in a kitchen environment (Narayanan, Aine, and Likhachev 2015), shown in Fig. 5. Specifically, the planner controls the position and orientation (x, y, θ) of the robot's base, the height of the prismatic spine which raises and lowers the torso, the 6 DoF pose of the gripper in the robot's body frame $(x_{hand}, y_{hand}, z_{hand}, roll_{hand}, pitch_{hand}, yaw_{hand})$, and the arm's "free angle" (which way the elbow is pointing).

Each vertex in the graph corresponds to a 11-dimensional robot configuration, while edges correspond to small kinematically feasible motion primitives that the robot can execute (for e.g., one motion primitive changes $roll_{hand}$ by 4 degrees). These edges are valid (existent) only if the motion is collision-free with the environment—which can be verified through expensive collision checking. The cost on an edge is the time taken to execute the motion along that edge, based on nominal velocities for base and joint angle movements. The start state is fully specified in 11 DoF, while the goal state is underspecified as a 6 DoF gripper configuration to allow the robot to pick up the object from different (x, y, θ) base locations around the object. We use 16 fixed motion primitives, and 3 adaptive ones that allow the robot to tuck its arm, untuck its arm, and snap the end-effector to the goal end-effector pose when close to the goal region.

We compute the priors on edge existence as follows. Let R_c denote the circumscribed radius of the robot at its fully outstretched configuration and $d(x, y)$ denote the 2D distance from the (x, y) location to the closest obstacle. Let $\text{base}(s)$ and $\text{ee}(s)$ represent the 2D locations of the base and end-effector corresponding to the full robot state s. Define $p(x, y) = \min(d(x, y)/R_c, 1.0)$ and $p(s) = \min(p(\text{base}(s)), p(\text{ee}(s)))$. Finally, the probability of existence for edge (s, s') is given by: $p(s, s') = \min(p(s), p(s'))$. The estimate of edge-evaluation times was set to be an identical constant for all edges. It is future work to develop and integrate more sophisticated models of edge-evaluation times.

For the search module, we use Focal-MHA* (Narayanan, Aine, and Likhachev 2015) on the augmented graph with three heuristics described in the same: a weighted version of the admissible heuristic, a base-heuristic to guide the robot to a location "behind" the goal with the correct gripper orientation, and another one to guide the base to the same location but with a tucked-arm configuration. We also introduce an additional probability heuristic to guide the robot along the most likely path to goal, which is computed by running a 2D Dijkstra search outward from the goal on the probability map $p(x, y)$. Additionally, we prioritize expanding from the probability heuristic queue[3] when it is making progress (measurable by monitoring h-values) rather than uniformly alternating between the heuristics.

For evaluation, we generated 100 random trials in which the tables are positioned differently every 10 trials, in addition to randomizing the clutter on the tables. For each trial, we choose a random staring configuration (11 DoF) for the robot, and a random pose (6 DoF) on one of the two tables for the gripper to reach. A trial is counted as successful if the planner returns a solution within a time limit of 2 minutes.

Table 1: Results for 11 DoF motion planning. Legend: A1: Probability Heuristic+AEE*, A2: Probability Heuristic+Lazy WA*, A3: Lazy WA* (all use Focal-MHA*). Base and arm costs are in meters and radians respectively. Plan times, expands and costs are averaged over instances on which all algorithms succeeded.

	Success (%)	Plan Time (s)	Expands	Base Cost	Arm Cost
A1	80	8.81	971.12	4.33	5.42
A2	77	12.79	1173.19	3.84	4.70
A3	65	10.22	1021.72	3.92	4.76

Table 1 compares AEE* and Lazy WA* (Cohen, Phillips, and Likhachev 2014) edge evaluation schemes under different configurations. Note that Lazy WA* is also edge equivalent to LazySP with the forward edge selector (Dellin and Srinivasa 2016). We use a suboptimality bound of $w = 100$ for all algorithms and report statistics only for the first solution found. The main observations are the significantly better success rates (solution found within time limit) for the methods which produce a diverse path-set (using the probability heuristic) for edge evaluation, and an improved performance for AEE* due of its probabilistic edge-existence reasoning.

6.2 Synthetic Benchmarking

We evaluate our algorithm on a synthetic 2D grid navigation domain with artificially inflated evaluation times for certain edges. Figure 6 shows the 2D map (with dimensions 564×507) used for the tests. The colored circles represent distinct probabilistic edge groups that may or may not exist in the graph. Further, these edges also have an artificially high evaluation time to verify their existence. To give an example of its relevance, imagine planning in (x, y) state space while doing edge evaluations through full 11-DoF planning that is more expensive. This abstraction is similar to the one used in BEAST (Kiesel and Ruml 2016), which focuses

[3]The probability queue also has restricted sharing—all other queues can expand states generated by every other queue, but the probability queue is only allowed to expand states it generated.

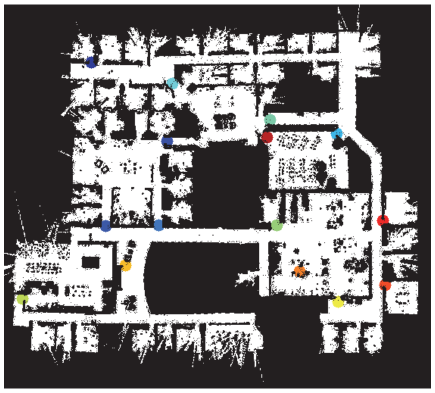

Figure 6: The 2D grid map used for benchmarking. The 15 colored circles represent regions (and consequently edges) in the grid that exist with some probability, and are also time-consuming to evaluate.

on online estimation of edge-existence probabilities, where edge evaluations are done using a sampling-based motion planner.

We compare ESP*+AEE* with Lazy WA* (Cohen, Phillips, and Likhachev 2014) with $W = 1$ (i.e., the optimal variant) under two control parameters: the time to evaluate existence of an edge group (T_e), and the probability of existence of an edge group (P_e). Both these parameters are held constant across all edge groups. We setup the problem such that the time to evaluate an edge in a specific group is large only for the first time, with successive evaluations of other edges in the group taking the same time as any deterministic edge. The motivation for this is twofold: first, this could represent spatial correlation between edges in the configuration space, and second, it simulates the abstraction described earlier in which an expensive edge-group evaluation represents solving a single subproblem. We chose Lazy WA* (with $W = 1$) for comparison since it was shown to perform consistently fast (with regard to total planning time) on different problems (Dellin and Srinivasa 2016) and also requires no expensive pre-computation. For both algorithms, we use a 4-connected grid and compute an admissible heuristic by running Dijkstra's search outward from the goal on the optimistic map (where all edge groups are assumed to exist). Both algorithms are guaranteed to eventually return the optimal solution if one exists, or report failure otherwise.

We run each algorithm on 100 trials (10 random samplings of the graph and 10 random start-goal pairs), for each parameter combination. Table 2 presents the average planning times and speedups obtained by Lazy ESP* over Lazy WA*. We use geometric means (GM) to present speedups, as they are better suited than the arithmetic mean (AM) when "averaging" ratios. For instance, two speedup ratios of 2.0 and 0.5 have a GM of 1 and an AM of 1.25. Since the algorithm was twice faster in one instance and twice slower in the other, a mean of 1 is more desirable than 1.25.

Readily noticeable is how ESP* dominates Lazy WA* for an edge-evaluation time of $500\ ms$ and vice versa for $10\ ms$. This is expected: when the evaluation times are small

Table 2: Comparison between Lazy WA* and our algorithm, ESP*+AEE* on the synthetic 2D grid navigation domain. Each data point is averaged over 100 trials (10 graph samplings $\times$ 10 random start-goal pairs) for the corresponding settings of edge-evaluation time (T_E) and edge existence probability (P_E). Legend: A1: ESP*+AEE*, A2: Lazy WA* $(W = 1)$. Speedup values are geometric means (arithmetic means in parentheses).

P_E		$T_E = 500\ ms$		$T_E = 100\ ms$		$T_E = 10\ ms$	
		A1	A2	A1	A2	A1	A2
0.75	Time (s)	1.95	2.27	0.97	0.49	0.72	0.08
	Speedup	1.78 (12.07)		0.83 (3.03)		0.18 (0.40)	
0.5	Time (s)	2.53	2.7	1.25	0.58	0.89	0.11
	Speedup	1.74 (13.5)		0.77 (2.45)		0.18 (0.38)	
0.25	Time (s)	1.83	2.44	0.96	0.53	0.71	0.10
	Speedup	2.22 (15.05)		0.91 (2.80)		0.20 (0.44)	

compared to the overhead of just searching (specifically on the augmented graph as done by ESP*), using a complicated path generator or edge-evaluation strategy only hurts. However, the moment edge evaluation becomes a critical bottleneck, using a sophisticated edge evaluation scheme does provide benefits. The results are more balanced for an evaluation time of $100\ ms$, implying that the fewer edge evaluations just about pay for the search overhead. Another interesting observation is how both methods take longer planning times with $P_E = 0.5$ compared to 0.25 or 0.75, in some sense confirming the intuition that it is easier to plan under low-entropy distributions.

7 Summary and Discussion

In this work, we presented a) a general strategy for interleaving planning and edge evaluation, b) a search algorithm (ESP*) for finding expected shortest paths on a graph with probabilistic edges, and c) a policy for edge evaluation (AEE*) given a set of candidate paths (possibly partial paths) with unevaluated edges and existence priors. We proved that ESP* can efficiently compute the expected shortest path cost, and that AEE* is optimal for edge evaluation in the anytime interruption sense. The experiments showed how the choice of edge-evaluation scheme is dependent on the problem setting at hand. Practically, we recommend using the ESP* + AEE* combination in domains where edge evaluations are expensive enough to justify the overhead of searching in the augmented graph. Typically, these tend to be domains where stochasticity is sparsely distributed, and where edge evaluation dominates the total planning time. In domains with many stochastic edges, alternative schemes for multiple path generation work well in conjunction with AEE*. Note that either of ESP* or AEE* could be used independently of the other, with a suitable complementary algorithm. For example, one could generate a set of candidate paths (without collision checking) from a sampling-based planner and use AEE* in an interleaved fashion. In the future, we would like to develop a bounded suboptimal version of ESP* to improve its tractability in domains with several stochastic edges.

Acknowledgments

This work was supported by NSF grant IIS-1409549 and by ARL under the Robotics CTA program grant W911NF-10-2-0016. We would also like to thank the anonymous reviewers for their valuable feedback and suggestions.

References

Bohlin, R., and Kavraki, L. E. 2000. Path Planning using Lazy PRM. In *International Conference on Robotics and Automation (ICRA)*, volume 1, 521–528. IEEE.

Choudhury, S.; Dellin, C. M.; and Srinivasa, S. S. 2016. Pareto-Optimal Search over Configuration Space Beliefs for Anytime Motion Planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Cohen, B.; Phillips, M.; and Likhachev, M. 2014. Planning Single-arm Manipulations with N-Arm Robots. In *Proceedings of Robotics: Science and Systems (RSS)*.

Dellin, C. M., and Srinivasa, S. S. 2016. A Unifying Formalism for Shortest Path Problems with Expensive Edge Evaluations via Lazy Best-First Search over Paths with Edge Selectors. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-quality Policies for the Canadian Traveler's Problem. In *Third Annual Symposium on Combinatorial Search (SoCS)*.

Felner, A.; Goldenberg, M.; Sharon, G.; Stern, R.; Beja, T.; Sturtevant, N. R.; Schaeffer, J.; and Holte, R. 2012. Partial-Expansion A* with Selective Node Generation. In *National Conference on Artificial Intelligence (AAAI)*.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. volume 129, 35–62. Elsevier.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, 28–35.

Huh, J., and Lee, D. D. 2016. Learning High-dimensional Mixture Models for Fast Collision Detection in Rapidly-Exploring Random Trees. In *International Conference on Robotics and Automation (ICRA)*, 63–69. IEEE.

Kiesel, S., and Ruml, W. 2016. A Bayesian Effort Bias for Sampling-based Motion Planning. In *Planning and Robotics Workshop (ICAPS-PlanRob16)*.

Narayanan, V.; Aine, S.; and Likhachev, M. 2015. Improved Multi-Heuristic A* for Searching with Uncalibrated Heuristics. In *Eighth Annual Symposium on Combinatorial Search (SoCS)*.

Papadimitriou, C. H., and Yannakakis, M. 1991. Shortest Paths without a Map. *Theoretical Computer Science* 84(1):127–150.

Thayer, J. T.; Benton, J.; and Helmert, M. 2012. Better Parameter-Free Anytime Search by Minimizing Time Between Solutions. In *Fifth Annual Symposium on Combinatorial Search (SoCS)*, 120–128.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with Partial Expansion for Large Branching Factor Problems. In *National Conference on Artificial Intelligence (AAAI)*, 923–929.