In Fifth International Workshop on Algorithmic Foundations of Robotics (WAFR 2002), Nice, France, December 2002

Expected Shortest Paths for Landmark-Based Robot Navigation

Amy J. Briggs[1], Carrick Detweiler[1], Daniel Scharstein[1], and Alexander Vandenberg-Rodes[2]

[1] Middlebury College, Middlebury VT 05753, USA
[2] Princeton University, Princeton NJ 08544, USA

Abstract. In this paper we address the problem of planning reliable landmark-based robot navigation strategies in the presence of significant sensor uncertainty. The navigation environments are modeled with directed weighted graphs in which edges can be traversed with given probabilities. To construct robust and efficient navigation plans, we compute *expected shortest paths* in such graphs. We formulate the expected shortest paths problem as a Markov decision process and provide two algorithms for its solution. We demonstrate the practicality of our approach using an extensive experimental analysis using graphs with varying sizes and parameters.

1 Introduction

Reliable strategies for mobile robot navigation using visual landmarks must be robust in the face of significant uncertainty in landmark detection. Varying lighting conditions, temporary occlusion, and unreliability of landmark recognition are all factors that contribute to such uncertainty, whether the visual landmarks are artificial or extracted from the environment. This paper presents algorithms and experimental results for computing expected shortest paths for use in navigation between landmarks. The paper builds on our real-time landmark detection system [16] and on our vision-based navigation framework [3,4]. Here we extend our prior work with significant contributions to the understanding of probabilistic navigation. First, we present two algorithms for the solution to the expected shortest paths problem and prove their convergence. Second, we present a careful analysis of the properties of the two algorithms, with an extensive empirical evaluation of their performance in practice. We first review the natural graph formulation of the problem, and then show how the problem can be concisely described as a Markov decision process.

The specific problem we address is the following. We assume an unmapped indoor environment (such as a factory floor or hospital) in which a mobile robot must repeatedly navigate. To enable the robot to localize itself quickly and then navigate in this space, the environment is enhanced with artificial visual landmarks. During an initial exploratory phase the robot systematically visits all landmarks and constructs their visibility graph. This graph is continually updated during subsequent navigation. All motion paths are

planned from landmark to landmark, moving along the visibility edges of the graph. Because landmark detection is unreliable, each edge is annotated not only with its length, but also with an estimate of the *probability* that the target landmark can be detected. These estimates are based on the history of all observations made by the robot, and are updated as the robot navigates the environment. Given such a graph, planning an optimal path to a goal landmark involves computing the *expected shortest path* (ESP), i.e., the path with expected shortest length. Using probabilistic techniques, our navigation system constructs reliable and efficient motion paths while compensating for occlusion, unreliability of landmark detection, and variations in visibility due to changes in the environment over time.

The ESP algorithms can be applied to important problems in other domains as well. For example, probabilistic graphs can be used to model communication networks in which individual links can fail. The ESP algorithms could then be used for optimal package routing given continually changing models of link reliability. Another application is that of traffic planning in a road network where some roads may be closed due to poor weather, accidents, congestion, road construction, or other temporary interruptions (e.g., draw bridges or railway crossings). The ESP formulation can also be applied to other robot path planning problems, for example to configuration-space planning in uncertain or changing environments.

After reviewing related work in Section 2, we define the expected shortest path problem and give two algorithms for its solution in Section 3. We then present the Markov decision process formulation of the problem and analyze the properties of our algorithms in Section 4. We report on our experimental results evaluating the performance of the algorithms on large sets of graphs in Section 5. We conclude with ideas for future work in Section 6.

2 Related work

Techniques for mobile robot navigation based on landmarks include those planned within a geometric environment map enhanced with perceptual landmarks [7,10], and those based on a topological description of landmark locations without a global map [12,17,23]. Many research groups have used artificial landmarks that can be easily and unobtrusively added to the environment for localization and navigation tasks [1,8,20].

Several different approaches to probabilistic path planning have been developed. Kavraki and Latombe [6] propose a randomized method for configuration space preprocessing that generates a network of collision-free configurations in a known environment. Overmars and Švestka [11] describe a similar probabilistic learning approach that extends to a number of motion planning problems. Several researchers have used partially observable MDPs for robot localization and navigation [18,19,21,23]. These approaches model the robot's state with a probability distribution, as opposed to our method,

which assumes a known state. However, POMDPs are computationally intractable and require approximate solutions such as a discretization of the state space [19] or "coastal navigation" [21].

Probabilistic graphs in which each edge is passable with a given probability have also been considered [9,13,14,22]. Most existing work, however, assumes a static graph, i.e., that unpassable edges do not become passable and vice versa. A typical problem in this scenario is to compute the probability that the graph is connected [14,22]. Mani et al. [9] study the problem of finding shortest paths in such static graphs, and focus on series-parallel graphs. Besides length and probability, they also associate a detection cost with each edge. Finally, Blei and Kaelbling [2] describe Markov decision processes for finding shortest paths in stochastic graphs with partially unknown topologies. Their work differs from ours in that they assume that an edge is either passable or not, but that the state of each edge is only known with a certain probability.

3 The expected shortest paths problem

We assume an environment augmented with N visual landmarks $a, b, c, \ldots$ that can be detected by the robot, albeit unreliably. We assume an edge from landmark a to landmark b has associated probability $p_{ab} \in [0, 1]$ and length $l_{ab} > 0$. The probability p_{ab} represents the likelihood that landmark b can be detected from landmark a; the length l_{ab} represents a measure of cost to traverse the edge (e.g., time of travel). We assume that the robot can only travel to landmarks that are currently visible. Thus, the probabilities p_{ab} more generally represent the likelihoods that edges are passable.

The robot, at any given landmark (or node) n, must choose among the currently visible landmarks (i.e., passable edges) where to travel to next. It also has the option of staying at the current node, and waiting for new landmarks to become visible. If no edge is currently passable, this is the only option. We represent the possibility of staying with a self-edge at each node n. Since staying is always an option, the associated probability is $p_{nn} = 1$. To prevent the robot from staying at a landmark indefinitely, we associate with the self-edge a non-zero cost $l_{nn} > 0$, for example, the time it takes to acquire a new image and to search it for landmarks.

A navigation task in such a *probabilistic graph* consists of a designated goal node g that the robot wants to reach from its current node s. In a non-probabilistic graph, the optimal path could be computed easily, for example using Dijkstra's shortest-path algorithm. In our case, however, we have to change the notion of a shortest path to that of a *path with shortest expected length*, or *expected shortest path* (ESP).

In previous work we have described a system in which a robot explores and navigates an unknown environment augmented with artificial visual landmarks, and constructs and updates a probabilistic graph in the process [3].

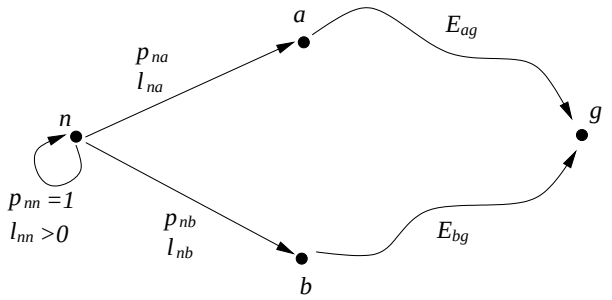

Fig. 1. A node n with self-edge and two outgoing edges to nodes a and b. All edges have lengths l_{ij} and probabilities of being passable p_{ij}. We wish to compute the expected lengths E_{ig} of the shortest paths from each node i to goal g. The four possible visibility scenarios at node n are reflected in Equation 2

Edge probabilities are estimated based on the history of observations made at each landmark.

Here we focus on the ESP problem and on the theoretical and practical properties of two algorithms for its solution. As mentioned earlier, the ESP problem and algorithms have applications beyond that of visual robot navigation and are important in their own right. We thus assume that a probabilistic graph as described above is given as input.

3.1 The ESP equations

Given a goal node g, the ESP problem is to compute for each node n the expected length E_{ng} of the shortest path from n to g. Clearly,

$$E_{gg} = 0. \tag{1}$$

We now derive equations that recursively relate the $N-1$ unknowns $E_{ng},\ n \neq g$.

Consider first a node n with two outgoing edges $n \rightarrow a$ and $n \rightarrow b$ (see Figure 1). The robot, upon arriving at n, will be faced with one of four *visibility scenarios*, depending on which of a and b can be detected. Suppose for example that a is visible but b is not. This scenario occurs with probability $p_{na}\,\overline{p_{nb}}$, where $\overline{p}$ denotes $(1-p)$. Given this situation, the robot has two options: to go to a, or to stay at n. (The latter may be desirable if b is usually visible and yields a much shorter path.) The expected total length of the path through node a is $l_{na} + E_{ag}$, while the expected total length of staying at node n is $l_{nn} + E_{ng}$. For notational convenience, let

$$L^i_{ng} = l_{ni} + E_{ig}$$

denote the total expected length of the path from n to g through i. The length of the shortest expected path (with b not visible) is then $\min(L^a_{ng},\ L^n_{ng})$, i.e., the smaller of the two candidate lengths. We can now write the complete

equation for E_{ng} by weighting the shortest candidate path with its corresponding probability in each of the four visibility scenarios:

$$\begin{aligned} E_{ng} = \quad & \overline{p_{na}}\ \overline{p_{nb}}\ L^n_{ng} \\ & + p_{na}\ \overline{p_{nb}}\ \min(L^a_{ng},\ L^n_{ng}) \\ & + \overline{p_{na}}\ p_{nb}\ \min(L^b_{ng},\ L^n_{ng}) \\ & + p_{na}\ p_{nb}\ \min(L^a_{ng},\ L^b_{ng},\ L^n_{ng}). \end{aligned} \tag{2}$$

It is easy to see how this equation generalizes to nodes with more than two outgoing edges. In particular, a node with k outgoing edges $n \to a,\ n \to b,\ \ldots,\ n \to z$ yields an equation with 2^k terms:

$$\begin{aligned} E_{ng} = \quad & \overline{p_{na}}\ \overline{p_{nb}} \ldots \overline{p_{nz}}\ L^n_{ng} \\ & + p_{na}\ \overline{p_{nb}} \ldots \overline{p_{nz}}\ \min(L^a_{ng},\ L^n_{ng}) \\ & + \overline{p_{na}}\ p_{nb} \ldots \overline{p_{nz}}\ \min(L^b_{ng},\ L^n_{ng}) \\ & + p_{na}\ p_{nb} \ldots \overline{p_{nz}}\ \min(L^a_{ng},\ L^b_{ng},\ L^n_{ng}) \\ & + \ldots \\ & + p_{na}\ p_{nb} \ldots p_{nz}\ \min(L^a_{ng},\ L^b_{ng}, \ldots, L^z_{ng},\ L^n_{ng}). \end{aligned} \tag{3}$$

We thus have a system of $N-1$ equations that recursively relate the $N-1$ unknowns $E_{ng},\ n \neq g$. Because of the minimum expressions the system cannot be solved directly. Note, however, that Equation 3 could be written as a linear equation with only k terms if the *ordering* among the candidate lengths L^i_{ng} were known. This observation plays an important role in our algorithms, so we will elaborate a bit:

Going back to the simpler Equation 2, consider the possible orderings among L^a_{ng}, L^b_{ng}, and L^n_{ng}. While one can deduce that L^n_{ng} cannot be the smallest of the three, each of the remaining four orderings is possible:

$$\begin{aligned} & L^a_{ng} \le L^b_{ng} \le L^n_{ng}, \qquad && L^a_{ng} \le L^n_{ng} \le L^b_{ng}, \\ & L^b_{ng} \le L^a_{ng} \le L^n_{ng}, \qquad && L^b_{ng} \le L^n_{ng} \le L^a_{ng}. \end{aligned} \tag{4}$$

Assume for example that the first of these orderings holds: $L^a_{ng} \le L^b_{ng} \le L^n_{ng}$. Such an ordering translates literally into a navigation strategy for a robot at landmark n: go to landmark a if possible (visible); if not, try to go to b if possible; else remain at n. Given this ordering, Equation 2 simplifies to

$$E_{ng} = p_{na}\ L^a_{ng} + \overline{p_{na}}\ p_{nb}\ L^b_{ng} + \overline{p_{na}}\ \overline{p_{nb}}\ L^n_{ng}. \tag{5}$$

A final observation: if L^n_{ng} is not last in the ordering, the subsequent terms in the equation drop out since $p_{nn} = 1$ (the self-edge can always be taken) and thus $\overline{p_{nn}} = 0$. For example, given the ordering $L^b_{ng} \le L^n_{ng} \le L^a_{ng}$, Equation 2 simplifies to:

$$E_{ng} = p_{nb}\ L^b_{ng} + \overline{p_{nb}}\ L^n_{ng}. \tag{6}$$

3.2 Algorithms for the ESP problem

To solve the ESP problem, we have developed variants of two algorithms commonly used to solve Markov decision processes (MDPs) (see Section 4 below). Borrowing the names from the MDP literature, these are *value iteration* (VI) and *policy iteration* (PI). Here we describe our algorithms in terms of the above notation. We then analyze some of their properties in the context of an MDP formulation in the next section.

Value iteration Recall that our goal is to solve the system of $N-1$ equations, each of the form of Equation 3. Collecting the $N-1$ unknowns E_{ng}, $n \neq g$, into a vector v we can write the entire system of equations concisely:

$$v = F(v). \tag{7}$$

Thus, the desired solution v of this system of equations is a *fixed point* of function F. The value iteration algorithm finds this fixed point by treating Equation 7 as an iterative process

$$v^{(n+1)} = F(v^{(n)}). \tag{8}$$

In [3] we have shown that this process converges to a unique fixed point v^* given an initial value $v^{(0)} = 0$, if there exists a path with non-zero probabilities from each node to the goal. Convergence is geometric once the current value is in the vicinity of the fixed point.

For an efficient implementation, we use the observation about orderings made in the previous section. That is, to evaluate Equation 3 for a node n with out-degree k, we first sort the candidate lengths L^i_{ng} in $O(k \log k)$ time, and then evaluate the resulting linear equation in $O(k)$ time. The total time for one iteration of Equation 8 is thus almost linear in the total number of edges in the graph E.

Policy iteration The idea of the second algorithm is to hypothesize edge orderings (as in Equation 4) at all nodes, and to *solve* the resulting linear system of Equation 7. The process is then repeated, using the previous solution to determine the new edge orderings. The iteration terminates when the solution to the current system of equations yields the same orderings as the previous solution. The name of the algorithm, policy iteration, reflects that instead of iterating over the *values* of the expected lengths, we iterate over the *ordering* of the expected lengths of the outgoing paths at each node. Recall that these orderings represent strategies (or policies) for the robot when faced with different visibility scenarios at a node.

Note that while each iteration of the VI algorithm requires evaluating $N-1$ linear equations (which together with the sorting step takes $O(N^2 \log N)$ time for dense graphs), each iteration of the PI algorithm requires *solving* this linear system, requiring a total time of $O(N^3)$ for dense graphs. As we show

in Section 5, however, the PI algorithm is competitive since it converges in very few iterations.

Analyzing the convergence properties of the policy iteration algorithm is easiest in the context of a Markov decision process (MDP) formulation, which we present next.

4 MDP formulation

The ESP problem can be formulated nicely as a Markov decision process (MDP). In brief, a MDP consists of a set of states S, and a set of allowable actions A_s for each state $s \in S$. Each action $\alpha \in A_s$ taken in state s yields a reward $r(s, \alpha)$, and results in a new (random) state s' according to a transition probability distribution $p(\cdot|s, \alpha)$. The objective is to devise a *policy* with a stationary *decision rule* $\delta : S \to A_s$ that selects a certain (fixed) action in each state so as to optimize a function of the total reward. This brief discussion ignores many common variations of MDPs, including time-dependent or discounted rewards, and non-stationary policies. For a comprehensive introduction to Markov decision processes, see the book by Puterman [15].

The ESP problem specifically translates into a non-discounted negative expected total-reward MDP. This means that each reward is interpreted as cost or penalty, and that the objective is to minimize the total expected cost. Upon reaching the goal g, no further cost is incurred.

In our case, the set of states S is the collection of landmarks. The concept of an allowable action at a landmark is slightly more complicated: it is not simply a destination landmark the robot should go to (which may not always be visible), but rather a *strategy* telling the robot what to do in any visibility scenario. Of course, such strategies correspond to the familiar edge orderings from Equation 4. Thus, the set of allowable actions A_s at landmark s is the set of all orderings among the outgoing edges of s.

For example, suppose the robot is at a node s with three outgoing edges $s \to a, s \to b, s \to c$. Then an example of $\alpha \in A_s$ would be: go to landmark c if possible; otherwise, go to landmark a if possible; otherwise, remain at s (take the self-edge) and try again. We notate this as $\alpha = s\colon c, a, s$. A robot following this action (strategy) will end up at c, a, or s, depending on the visibility scenario. The transition probabilities are thus

$$\begin{aligned} p(c|s,\alpha) &= p_{sc} \\ p(a|s,\alpha) &= \overline{p_{sc}}\, p_{sa} \\ p(s|s,\alpha) &= \overline{p_{sc}}\, \overline{p_{sa}}. \end{aligned} \tag{9}$$

The expected cost for α is simply the expected length for this action

$$r(s,\alpha) = p_{sc}\, l_{sc} + \overline{p_{sc}}\, p_{sa}\, l_{sa} + \overline{p_{sc}}\, \overline{p_{sa}}\, l_{ss}. \tag{10}$$

Note that $p(i|s,\alpha) = 0$ for any other landmark i since the self-edge can always be taken. We can therefore write the expected cost for action α as

$$r(s,\alpha) = \sum_{i \in S} l_{si}\; p(i|s,\alpha). \tag{11}$$

A decision rule (or policy) δ assigns to each landmark one action (strategy) α. Under a given policy, the robot traverses the graph until it reaches the goal landmark. We assume the action at goal g is always $\alpha = g : g$, and $r(g,\alpha) = 0$. Since we wish to find the expected shortest path, we seek a policy that minimizes the total expected cost, i.e., the sum of the costs accrued by the actions leading to the goal.

We now derive the total expected cost for a given decision rule δ. Let us denote the landmarks $x_1, \ldots, x_N$. Let r_δ be the vector containing the expected cost for the action given by δ at each of the N landmarks. That is, entry s of r_δ is

$$r_\delta[s] = r(s, \delta(x_s)). \tag{12}$$

Let P_δ be the $N \times N$ transition probability matrix under the decision rule δ, i.e., the entry at row s and column d is

$$P_\delta[s,d] = p(x_d|x_s, \delta(x_s)). \tag{13}$$

Now define the j-step transition probability matrix ${}_jP_\delta$ such that ${}_jP_\delta[s,d]$ is the probability of arriving at x_d after j transitions from x_s. The probabilities after $j+1$ steps are then

$$\begin{aligned} {}_{j+1}P_\delta[s,d] &= \sum_{i=1}^{N} {}_jP_\delta[s,i]\; p(x_d|x_i, \delta(x_i)) \\ &= \sum_{i=1}^{N} {}_jP_\delta[s,i]\; P_\delta[i,d], \end{aligned} \tag{14}$$

and therefore ${}_{j+1}P_\delta = {}_jP_\delta\, P_\delta$. Since ${}_1P_\delta = P_\delta$, we get by induction

$${}_jP_\delta = P_\delta^j. \tag{15}$$

The vector of expected costs for the first transition is just r_δ. It is easy to see that the costs for the second transition are $P_\delta\, r_\delta$; for the third, $P_\delta^2\, r_\delta$, etc. The vector v_δ of total expected costs for δ is thus the infinite sum

$$v_\delta = \sum_{i=0}^{\infty} P_\delta^i\, r_\delta. \tag{16}$$

Expanding this sum we get

$$\begin{aligned} v_\delta &= r_\delta + P_\delta\, r_\delta + P_\delta^2\, r_\delta + P_\delta^3\, r_\delta + \ldots \\ &= r_\delta + P_\delta(r_\delta + P_\delta\, r_\delta + P_\delta^2\, r_\delta + \ldots), \end{aligned}$$

and thus

$$v_\delta = r_\delta + P_\delta \, v_\delta. \tag{17}$$

So, given a decision rule δ we can compute the expected lengths by solving

$$(I - P_\delta) \, v_\delta = r_\delta. \tag{18}$$

We now ask the question "which policy results in the shortest total expected lengths?" and present our two algorithms, value iteration and policy iteration, in this MDP formulation. The difference between the two is that VI iterates over the expected lengths v, while PI iterates over the policies δ.

4.1 Value iteration

The value iteration algorithm computes a sequence of values $v^{(0)}, v^{(1)}, \ldots$, as follows:

1. Choose an initial $v^{(0)}$, and set $n = 0$.
2. Compute
$$v^{(n+1)}[k] = \min_{\alpha \in A_{x_k}} \left(r(x_k, \alpha) + \sum_{i=1}^{N} p(x_i | x_k, \alpha) \, v^{(n)}[i] \right). \tag{19}$$
3. If $v^{(n+1)}$ is very close to $v^{(n)}$, then terminate and return the policy δ consisting of the minimizing actions α for all nodes. Else, increment n by 1, and go to step 2.

When written in matrix form, Equation 19 becomes

$$v^{(n+1)} = \min_\delta (r_\delta + P_\delta v^{(n)}), \tag{20}$$

which parallels Equation 8 in Section 3.2. Our convergence result from [3] guarantees that Equation 20 has a unique fixed point v^*, and that the sequence $v^{(n)}$ converges monotonically to v^* when $v^{(0)} = 0$.

4.2 Policy iteration

The policy iteration algorithm computes a sequence of policies $\delta^{(0)}, \delta^{(1)}, \ldots$, as follows:

1. Choose an initial policy $\delta^{(0)}$, and set $n = 0$.
2. Solve $(I - P_{\delta^{(n)}}) \, v^{(n)} = r_{\delta^{(n)}}$ for $v^{(n)}$.
3. If $\delta^{(n)}$ is in the set
$$D = \arg\min_\delta (r_\delta + P_\delta v^{(n)}),$$
then terminate and return $\delta^{(n)}$. Else choose some $\delta^{(n+1)} \in D$, increment n by 1, and go to step 2.

We now turn to the convergence proof. It turns out that for general non-discounted negative expected reward MDPs, the policy iteration algorithm does *not* always converge to the optimal policy [15]. Given the conditions in our model, however, it does.

We will first prove the following lemma, analog to Proposition 7.3.13 in the text by Puterman [15].

Lemma 1 *For each iteration n in policy iteration, $v^{(n+1)} \leq v^{(n)}$.*

Proof: By step 3 in the policy iteration algorithm, if $\delta^{(n)}$ is in the set D, then $v^{(n+1)} = v^{(n)}$. If $\delta^{(n)}$ is not in D, then

$$r_{\delta^{(n+1)}} + P_{\delta^{(n+1)}} v^{(n)} \leq r_{\delta^{(n)}} + P_{\delta^{(n)}} v^{(n)} = v^{(n)}. \tag{21}$$

Because all entries of $P^k_{\delta^{(n)}}$ are non-negative for every $k > 0$, we can substitute $r_{\delta^{(n+1)}} + P_{\delta^{(n+1)}} v^{(n)}$ for $v^{(n)}$ in the above inequality, giving:

$$r_{\delta^{(n+1)}} + P_{\delta^{(n+1)}}(r_{\delta^{(n+1)}} + P_{\delta^{(n+1)}} v^{(n)}) \tag{22}$$

$$= (I + P_{\delta^{(n+1)}}) r_{\delta^{(n+1)}} + P^2_{\delta^{(n+1)}} v^{(n)} \leq v^{(n)}. \tag{23}$$

So by induction,

$$(\sum_{k=0}^{K-1} P^k_{\delta^{(n+1)}} r_{\delta^{(n+1)}}) + P^K_{\delta^{(n+1)}} v^{(n)} \leq v^{(n)} \tag{24}$$

for any positive K. Since all entries of $v^{(n)}$ are non-negative,

$$\sum_{k=0}^{K-1} P^k_{\delta^{(n+1)}} r_{\delta^{(n+1)}} \leq v^{(n)} \tag{25}$$

for $K > 0$. Taking the limit as K goes to infinity gives

$$\sum_{k=0}^{\infty} P^k_{\delta^{(n+1)}} r_{\delta^{(n+1)}} \leq v^{(n)}. \tag{26}$$

Therefore by Equation 16, $v^{(n+1)} \leq v^{(n)}$. □

We are now prepared to prove our main theorem:

Theorem 1. *Given the above model and an initial policy $\delta^{(0)}$ such that $v_{\delta^{(0)}}$ is finite, policy iteration converges to a policy δ with minimal v_δ.*

Proof: Suppose we start with some $\delta^{(0)}$ such that $v^{(0)}$ is finite. By Lemma 1, for each iteration n, $v^{(n+1)} \leq v^{(n)}$. There are finitely many edges in the graph, so there are only finitely many decision rules (since each decision rule is an ordering of the edges). Since the sequence $v^{(n)}$ is non-increasing and no decision rule appears twice, the algorithm must terminate, because there exists an optimal decision rule. When it does at the n-th iteration, $\delta^{(n)}$ is in $\arg\min(r_\delta + P_\delta v^{(n)})$, so $v^{(n)} = \min(r_\delta + P_\delta v^{(n)})$. Our fixed-point result from

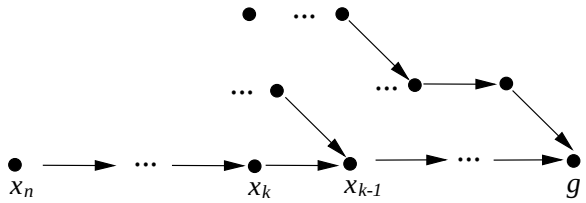

Fig. 2. Every landmark x_n is part of a chain of landmarks that leads to goal g

the VI algorithm then guarantees that we have the optimal decision rule δ. □

The remaining problem is to find an initial policy $\delta^{(0)}$ that yields finite expected costs $v_{\delta^{(0)}}$. Note that there are many policies that yield infinite expected costs. Intuitively, such policies can "trap" the robot within a subgraph from which it cannot reach the goal. The simplest example is a policy containing an action $\alpha = s\colon s$ that commands the robot to always stay at a node s. Circular traps such $s\colon t, s$ and $t\colon s, t$ are also possible.

A policy δ with finite v_δ can easily be constructed, however, using a breadth-first search that starts at the goal node g and follows the edges in reverse direction. Initially, the action $\alpha = s\colon g, s$ is assigned to each node s that has a direct edge to the goal. Recursively, actions $\alpha = s\colon b(s), s$ are then assigned to other nodes s, where $b(s)$ denotes the node's predecessor in the search (i.e., a node one step closer to the goal).

The above algorithm performs a backwards search through every edge, and thus takes $O(E)$ time, where E is the number of edges in the graph. It assigns actions that impose a (non-circular) tree structure on the graph, with goal g forming the root of the tree (see Figure 2). In particular, each node x_n is part of a chain of nodes leading to the goal

$$x_n \to \ldots \to x_k \to x_{k-1} \to \ldots \to g.$$

Assume x_k and x_{k-1} are two successive nodes along that chain. By Equation 17,

$$v[k] = r(x_k, \alpha) + p_{x_k x_{k-1}} v[k-1] + \overline{p_{x_k x_{k-1}}}\, v[k], \tag{27}$$

so

$$v[k] = \frac{r(x_k, \alpha)}{p_{x_k x_{k-1}}} + v[k-1]. \tag{28}$$

Since the probabilities for each landmark are non-zero, and the costs are finite, by induction from the goal landmark g, each $v[k]$ through $v[n]$ is finite. Therefore the above breadth-first search algorithm gives a suitable starting policy $\delta^{(0)}$ for the policy iteration algorithm.

We now have convergence proofs for both VI and PI algorithms, and also an algorithm for constructing an initial policy for PI. To our knowledge,

no theoretical results exist about the convergence speed of either algorithm. Thus, in order to assess the practicality of both VI and PI algorithms, we now turn to our empirical evaluation.

5 Experiments

We have implemented both the value iteration and policy iteration algorithms, and have performed an extensive experimental evaluation to assess their respective performance. Our implementation is in C, and we use Matlab for solving the linear system of equations in the policy iteration algorithm.

As a first step in both algorithms we compute the actual shortest paths (ignoring the probabilities) using Dijkstra's shortest path algorithm. This is done for several reasons: First, in a graph that is being constructed by a robot exploring the environment, the goal may not be reachable from all nodes (e.g., from nodes that the robot has only seen but not yet visited). Running Dijkstra's algorithm will identify these nodes, which can then be removed before solving the ESP problem. Second, the lengths of the actual shortest paths are lower bounds on the lengths of the expected shortest paths, and can be used as a better initial estimate for the unknowns E_{ng} for the value iteration algorithm. (The VI convergence proof is based on bounded increasing sequences and extends to any start value that is component-wise smaller than the solution.) Finally, for policy iteration, it turns out that the actual shortest paths provide an alternate way of finding an initial policy that yields finite expected lengths, and thus a solvable system. While the breadth-first search algorithm from Section 4.2 for finding an initial policy is asymptotically faster, extra memory and time are needed to establish back-pointers along all edges before the actual search, which are not required by Dijkstra's algorithm. In practice, we have found that the two methods have very similar running times.

For testing purposes we have generated more than 50,000 graphs with different properties, including number of nodes N, number of edges E, and range of probability values. Graphs used have up to 3000 nodes; graph densities range from sparse ($E \propto N$) to dense ($E \propto N^2$); and probability values are characterized as very low (0.0001–0.001), low (0.0001–0.5), or full (0.0001–1). We have also experimented with different graph structures, including large diameter "corridor" graphs, and "multi-room" graphs (sparsely connected collections of highly connected subgraphs, simulating the visibility graphs of multi-room buildings).

Here we report on a subset of our results. The plots shown in Figure 3 measure the number of iterations necessary for VI and PI as a function of the number of nodes N in the graph. Each plot shows the results of 1250 individual experiments on random graphs with up to 2500 nodes. We have performed such experiments for many different types of graphs and parameters. The plots in Figure 3 contrast sparse graphs ($E \approx N$) with dense graphs

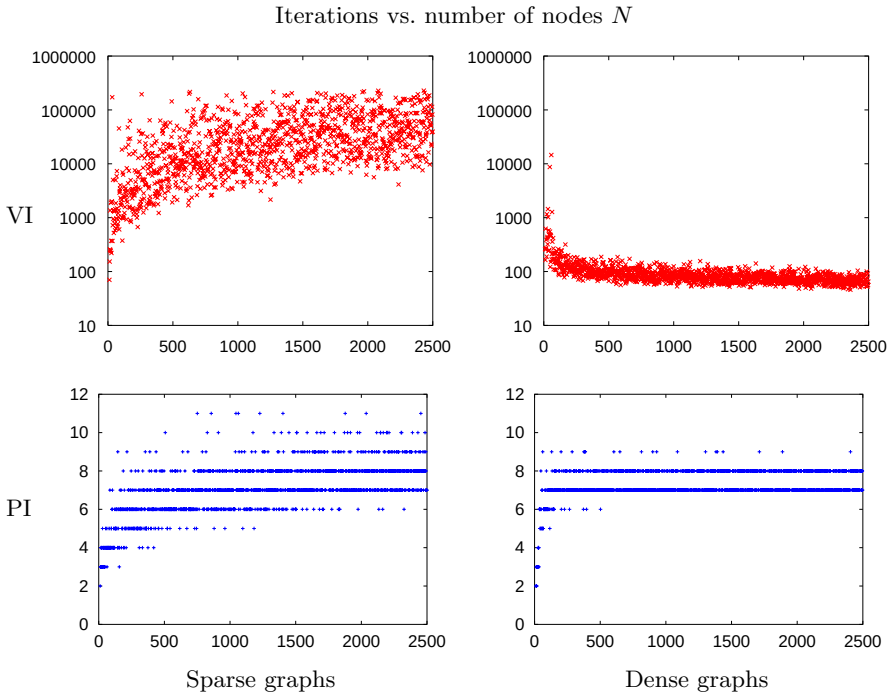

Fig. 3. Number of iterations for VI (first row) and PI (second row) algorithms as a function of the number of nodes in the graph. The plots on the left show results for sparse graphs ($E \approx N$), the plots on the right show results for dense graphs ($E \approx N^2/15$)

($E \approx N^2/15$), both with full random probabilities (ranging from 0.0001 to 1). Examining the plots in the first row of Figure 3, it can be seen that the value iteration algorithm requires large numbers of iterations (often upwards of 10,000) on sparse graphs, while it converges much faster on dense graphs (typically within a few hundred iterations). The likely reason for this is that changes in the variables can propagate much faster in densely connected graphs. Note that the initial "peak" in the second plot is due to our definition of "dense" as $E \approx N^2/15$, which for small N still results in fairly sparse graphs. Interestingly, the average number of iterations for dense graphs does not increase with larger values of N — if anything, there is a slight decrease.

Results for the policy iteration algorithm (shown in the second row of Figure 3) are vastly different. In all cases, PI only takes very few iterations to converge. While the number of iterations increases with the number of nodes, the curves flatten out quickly, and the distributions of iterations for $N = 1000$ are very similar to those for $N = 2500$. Even for the largest N, most sparse graphs take no more than 6–9 iterations to converge; the dense graphs rarely take more than 7 or 8 iterations. In fact, among the more than

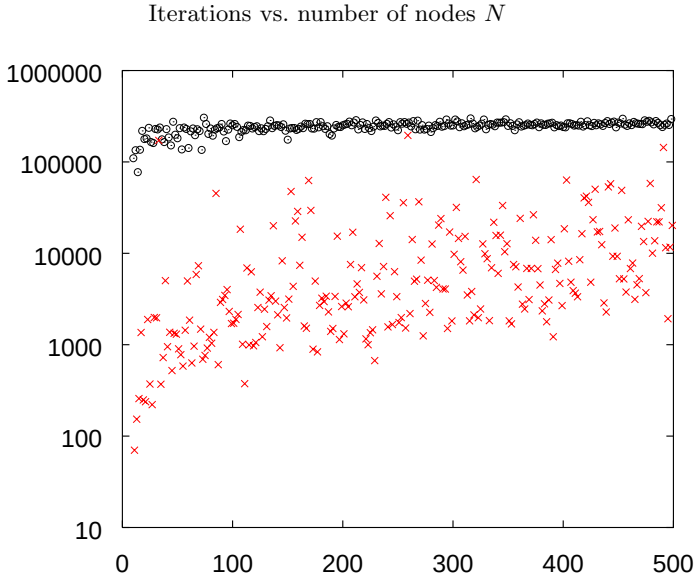

Fig. 4. Number of iterations for VI for normal (red ×) and very low (black ∘) edge probabilities on sparse graphs

50,000 graphs we have generated, only a handful require 11 iterations, only 3 require 12 iterations, and we have yet to find a graph that requires more than 12 iterations. As is the case for VI, the distribution of iterations for dense graphs is narrower and has fewer outliers.

We have also investigated the effect of different probability ranges. The number of iterations used by the PI algorithm remains roughly the same, although a low probability range (0.0001 to 0.5) often requires one extra iteration. The VI algorithm, however, is strongly affected by the probability ranges. This is demonstrated in Figure 4, which shows the number of iterations of the VI algorithm on sparse graphs for two different probability ranges: full (0.0001–1), and very low (0.0001–0.001).

The results of other experiments we have performed are consistent with the ones reported here. In particular, for PI, the curve defined by the number of iterations vs. graph size always resembles a logarithmic shape, indicating that few iterations are necessary even for very large graphs. Also, we have been unable to construct "pathological cases" that require a certain minimum number of iterations for PI. Value iteration, on the other hand, can be made to converge arbitrarily slowly by constructing graphs whose edges all have very low probabilities as shown in Figure 4. Although such graphs may not be common in practical applications, this is further evidence for the unpredictability of the convergence speed of VI that is also exhibited in the top left plot in Figure 3.

We now turn to the actual running times of the two algorithms. Figure 5 shows plots that compare the running times of VI and PI as a function of N on the same sets of graphs as in Figure 3. All experiments were performed under Linux on a 1.53 GHz Athlon machine with 512 MB of memory. It can be seen that policy iteration clearly outperforms value iteration, in particular on sparse graphs, where PI is faster by several orders of magnitude. Much of this performance gain is due to the fact that we use Matlab's sparse matrix representation and equation solver. (Using a regular solver for sparse graphs is slower by at least a factor of 10.) On dense graphs, the margin between PI and VI is much smaller, since VI takes fewer iterations than it does on sparse graphs, but PI's equation solving step requires more work. Even in this case, however, PI is the winner, running at least a factor of 2 faster than VI. It should be noted that the reported running times are process times. When we measure total time, the running times of PI and VI on dense graphs become almost identical. The likely reason for this is that the time for PI's memory allocation steps are not measured in the process time.

Graphs encountered in real environments span the range of the two types of graphs discussed here. In practical applications, sparse graphs will be more common, and PI, although slightly more difficult to implement, will be the better choice.

A practical issue is the memory requirements of the two algorithms. VI, which only needs to *evaluate* the system of equations, requires only enough memory to store the graph. PI, which needs to *solve* the system, needs up to $O(N^2)$ space to store the matrix representing this system. Given a large enough graph, the matrix will be too large to be allocated, making PI impossible to use. For our implementation and hardware, this happens for dense graphs upwards of about 2800 nodes. In such situations, VI would be the only choice. Since we are using Matlab's sparse matrix representation, however, the memory is sufficient for much larger sparse graphs. For example, a sparse graph with 15,000 nodes and about 25,000 edges can still be solved using PI in less than 1 second.

6 Conclusion

In this paper we have defined the *expected shortest path* (ESP) problem, which arises in landmark-based robot navigation where the reliability of landmark detection is modeled explicitly. We have presented two algorithms for its solution, *value iteration* (VI) and *policy iteration* (PI), and have proven their convergence properties in the context of a MDP formulation. Using an extensive experimental analysis, we have demonstrated the practicality of both algorithms, and have shown that PI usually outperforms VI. In particular, on sparse graphs, PI is orders of magnitudes faster. On very large dense graphs, however, PI cannot be used due to its larger memory requirements.

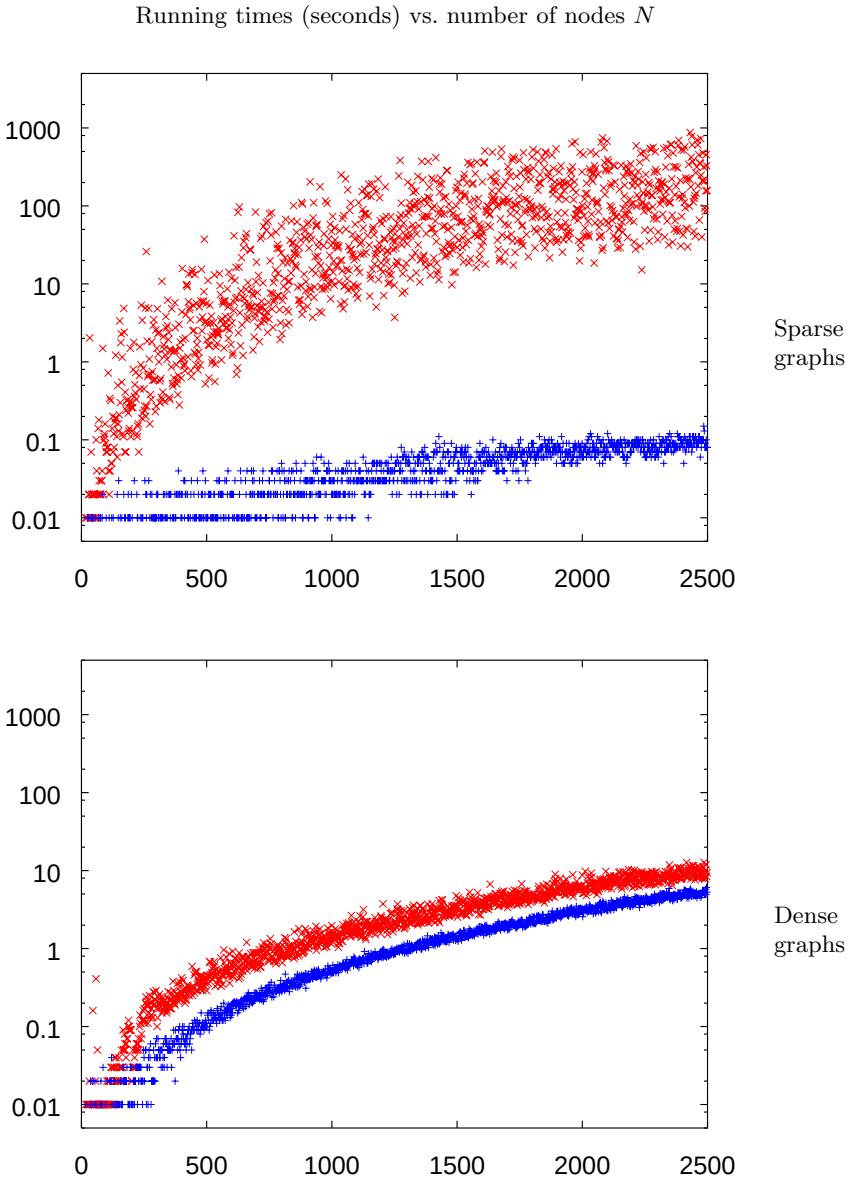

Fig. 5. Running times in seconds for VI (red $\times$) and PI (blue $+$) algorithms as a function of the number of nodes in the graph. The first plot shows results for sparse graphs ($E \approx N$); the second plot shows results for dense graphs ($E \approx N^2/15$)

In future work we plan to investigate the application of our algorithms to navigation using natural landmarks extracted from the environment. Building on related work in this area [5,10,12,17], we aim to extend our probabilistic framework to obtain more reliable and efficient path planners for vision-based robot navigation in arbitrary environments.

Acknowledgments

The authors are grateful for the insights of Steve Abbott and Deniz Sarioz that contributed to this work, and for the assistance of Darius Braziunas, Victor Dan, Cristian Dima, Huan Ding, David Ehringer, Dan Knights, Jeff Lanza, Fafa Paku, and Peter Wall in the implementation of the navigation framework presented here.

Support for this work was provided in part by the National Science Foundation under grants IIS-0118892, CCR-9902032, CAREER grant 9984485, POWRE grant EIA-9806108, by Middlebury College, by the Howard Hughes Medical Institute, and by the Council on Undergraduate Research.

References

1. C. Becker, J. Salas, K. Tokusei, and J.-C. Latombe. Reliable navigation using landmarks. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 401–406, June 1995.
2. D. M. Blei and L. P. Kaelbling. Shortest paths in a dynamic uncertain domain. In *Proceedings of the IJCAI Workshop on Adaptive Spatial Representations of Dynamic Environments*, 1999.
3. A. Briggs, D. Scharstein, and S. Abbott. Reliable mobile robot navigation from unreliable visual cues. In Donald, Lynch, and Rus, editors, *Algorithmic and Computational Robotics: New Directions, A. K. Peters*, pages 349–362, 2001.
4. A. Briggs, D. Scharstein, D. Braziunas, C. Dima, and P. Wall. Mobile robot navigation using self-similar landmarks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2000)*, pages 1428–1434, April 2000.
5. C. Fennema, A. Hanson, E. Riseman, J. R. Beveride, and R. Kumar. Model-directed mobile robot navigation. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6):1352–1369, 1990.
6. L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2138–2145, May 1994.
7. A. Lazanas and J.-C. Latombe. Landmark-based robot navigation. *Algorithmica*, 13(5):472–501, May 1995.
8. C. Lin and R. Tummala. Mobile robot navigation using artificial landmarks. *Journal of Robotic Systems*, 14(2):93–106, 1997.
9. M. Mani, A. Zelikovsky, G. Bhatia, and A. Kahng. Traversing probabilistic graphs. Technical Report 990010, UCLA, 1999.

10. B. Nickerson, P. Jasiobedzki, D. Wilkes, M. Jenkin, E. Milios, J. Tsotsos, A. Jepson, and O. N. Bains. The ARK project: Autonomous mobile robots for known industrial environments. *Robotics and Autonomous Systems*, 25:83–104, 1998.
11. M. H. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In Goldberg, Halperin, Latombe, and Wilson, editors, *1994 Workshop on the Algorithmic Foundations of Robotics, A. K. Peters*, pages 19–37, 1995.
12. C. Owen and U. Nehmzow. Landmark-based navigation for a mobile robot. In *Proceedings of Simulation of Adaptive Behaviour*. MIT Press, 1998.
13. C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84:127–150, 1991.
14. J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal of Computing*, 12(4), November 1983.
15. M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, NY, 1994.
16. D. Scharstein and A. Briggs. Real-time recognition of self-similar landmarks. *Image and Vision Computing*, 19(11):763–772, September 2001.
17. R. Sim and G. Dudek. Mobile robot localization from learned landmarks. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems (IROS), Victoria, BC*, October 1998.
18. R. Simmons, J. Fernandez, R. Goodwin, S. Koenig, and J. O'Sullivan. Xavier: An autonomous mobile robot on the web. *IEEE Robotics and Automation Magazine*, 1999.
19. R. Simmons and S. Koenig. Probabilistic navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1080–1087, 1995.
20. C. J. Taylor and D. J. Kriegman. Vision-based motion planning and exploration algorithms for mobile robots. In Goldberg, Halperin, Latombe, and Wilson, editors, *1994 Workshop on the Algorithmic Foundations of Robotics, A. K. Peters*, pages 69–83, 1995.
21. S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. Probabilistic algorithms and the interactive museum tour-guide robot Minerva. *International Journal of Robotics Research*, 19(11), 2000.
22. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8(3), August 1979.
23. F. Zanichelli. Topological maps and robust localization for autonomous navigation. In *Proceedings of the International Joint Conference on Artificial Intelligence, Workshop ROB-2*, 1999.