Utility-based On-Line Exploration for Repeated Navigation in an Embedded Graph [1]

Shlomo Argamon-Engelson [a,2], Sarit Kraus [a,b], Sigalit Sina [a]

[a] *Dept. Mathematics and Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel*
Email: {argamon,sarit,sina}@cs.biu.ac.il

[b] *Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742*

Abstract

In this paper, we address the tradeoff between exploration and exploitation for agents which need to learn more about the structure of their environment in order to perform more effectively. For example, a robot may need to learn the most efficient routes between important sites in its environment. We compare on-line and off-line exploration for a *repeated* task, where the agent is given some particular task to perform some number of times. Tasks are modeled as navigation on a graph embedded in the plane. This paper describes a *utility-based* on-line exploration algorithm for repeated tasks, which takes into account both the costs and potential benefits (over future task repetitions) of different exploratory actions. Exploration is performed in a greedy fashion, with the locally optimal exploratory action performed on each task repetition. We experimentally evaluated our utility-based on-line algorithm against a heuristic search algorithm for off-line exploration as well as a randomized on-line exploration algorithm. We found that for a single repeated task, utility-based on-line exploration consistently outperforms the alternatives, unless the number of task repetitions is very high. In addition, we extended the algorithms for the case of multiple repeated tasks, where the agent has a different randomly-chosen task to perform each time. Here too, we found that utility-based on-line exploration is often preferred.

[1] This research was supported in part by the National Science Foundation grant number IRI-9724937.
[2] Supported during part of this work by a fellowship from the Fulbright Foundation.

1 Introduction

Intelligent agents in the real world often have to perform tasks about which they have only incomplete knowledge. In particular, an agent may have only partial knowledge of the actions it can take and their effects. In such a case, it is important for the agent to learn about the structure of its environment, in order to better accomplish its goals. Such learning involves *exploration*, in which actions are chosen for the goal of increasing the agent's knowledge, as opposed to *exploitation* where actions are chosen which directly lead toward accomplishing the agent's given task. Exploring the world and learning its structure may be performed either in a separate exploration phase *off-line*, before performing any tasks, or *on-line*, while performing tasks.

In this paper, we address the tradeoff between exploration and exploitation for agents which need to learn more about the structure of their environment in order to perform more effectively. The need for such learning is ubiquitous; any agent needing to operate in the real world can have only partial knowledge of its environment, and therefore must be able to explore and learn. For example, a mobile robot may need to learn a map of its environment so that it can navigate effectively from place to place.

The environment model used in this paper is motivated mainly by the case of a mobile robot moving in the plane. We model the agent's environment as a graph embedded in the plane, each of whose nodes (with its position in the plane) represents a distinct place, and each of whose arcs represents an action which moves the agent from one place to another. Similar graph-based world models have long been used in theoretical work on off-line learning, such as [12,13,24,26]. These models have also been extended to include sensor and effector noise [2,10]. A task situated between off-line and on-line exploration, is piecemeal exploration [6], in which the robot must return to its 'home' every so often.

On-line state-space learning has been addressed by so-called 'real-time' search algorithms [8,17,22]. These algorithms typically achieve a single task (possibly moving) while learning the structure of the environment. In the worst case, these algorithms will explore the entire state-space, depending on the quality of the heuristic function. Improved efficiency in real-time search has been obtained by restricting the use of heuristic exploitation [27] and by not requiring the algorithm to find an optimal policy [18]. In more specific problem settings, more informed heuristics may be applied, for example Cucka et al. [9] use information about the geometric direction of the goal to heuristically improve the exploration process.

In this paper, we compare on-line and off-line exploration for a *repeated* task, where the agent is given some particular task(s) to perform some number of times. We assume that the agent has the ability to perform the task, even before acquiring specific knowledge about its environment; acquiring such knowledge through exploration may then allow the agent to perform more effectively.

With on-line exploration, performance of the task becomes more efficient over time, as the agent learns better ways to perform it [7]. As the number of repetitions gets very large, it might be preferable to learn the entire structure of the state-space first, so that the agent can be sure to use the most efficient plan for its task. In more realistic scenarios, this is not the case, since the number of repetitions of a particular task is bounded. In such a case, it may be more effective to learn on-line, and avoid wasting effort in exploring areas of the world which do not contribute to task performance.

We believe that deciding how to explore, like planning in general, should be performed in a decision-theoretic manner, accounting for the utility and cost of alternative courses of action [11,15,16]. We describe in this paper a *utility-based* on-line exploration algorithm for a partially known environment. In our approach, the agent estimates the expected utility of possible exploratory actions in order to decide whether and how to explore. This utility is evaluated relative to the agent's future tasks and the cost of exploring, taking into account the extra cost of exploring vs. just using the best plan currently known. For example, if the agent only has to perform a task once, it may not be worthwhile to explore at all, since the cost may be higher than the expected gain for that single task. The utility of exploration then increases with the number of future task repetitions. The algorithm chooses exploratory actions in a greedy fashion, in order to avoid exponential increase in the number of future courses of action that need to be considered (similar to Etzioni's [15] use of a greedy marginal utility heuristic).

Methods for solving Markov decision processes (MDPs) also involve an on-line tradeoff between exploration and exploitation [25]. In these problems, the environment is modeled as a probabilistic finite-state machine where the agent receives rewards for being in particular states. The transition from one state to another occurs with some probability depending on what action the agent takes from the first state. The agent's task is to maximize its total reward, learning something about the reward and transition probability distributions in the process (based on known priors over those distributions). Optimal solutions are known for this problem [3,5], as well as its simpler variant the k-armed bandit problem [4], but these solutions are of exponential complexity [20]. Various faster approximate strategies have also been proposed for this problem [19,23,28] and have been shown to be useful. These methods assume knowledge of the states and possible transitions in the MDP, and so are not directly applicable to our problem in this paper, since we do not assume that the agent knows anything at the start about the specific structure of its environment.

Our approach of incremental utility-based exploration may also be compared to the anytime algorithm of Dean et al. [11] for decision-theoretic planning in (completely known) stochastic environments. Their method creates an optimal policy for a small part of the environment (the *envelope*), and incrementally extends the envelope in order to increase the usefulness of the generated policy. Exploration methods such as that described in this paper could extend the usefulness of such planning techniques to incompletely known environments.

We experimentally evaluated utility-based on-line exploration on a variety of randomly generated environments, and compared the performance of our greedy utility-based on-line algorithm against that of a heuristic search algorithm for off-line exploration. Our results show that utility-based on-line exploration is nearly always preferable to off-line exploration. Furthermore, we investigated the contribution of the utility-based formulation, by comparing it to a randomized approach to on-line exploration (similar to methods commonly used in reinforcement learning [21,29]).

We found that for a single repeated task, utility-based on-line exploration usually outperforms the alternatives in our model, unless the number of task repetitions is very high. We also extended the algorithms to explore for multiple repeated tasks, where the agent has a different, randomly-chosen (from a known subset of possible tasks), task to perform each time. Surprisingly, we found that utility-based on-line exploration may be preferred to off-line exploration, even for large numbers of different tasks.

2 Problem Definition

We model repeated navigational tasks by the following problem of repeated navigation in a partially known graph embedded in the plane. An *environment* consists of a connected undirected graph $G = (V, E)$ where V is a set of N nodes and E is the set of edges of the graph, where each node v is assigned a position in the plane. When the agent is at node v, it can sense (with certainty) v's position as well as the direction of each edge adjoining v (each edge is assumed to be a straight path in the plane). The agent does not know the length of each edge, however, and so does not know where the edge ends.

The agent is given the following task. There are two *target nodes* in V, A and B, and the agent needs to go from A to B and back some given number of times, denoted by R. The agent starts out knowing only the positions of A and B. Until the agent starts moving, it does not know about any nodes other than A and B or about any of the edges in the environment. Our goal is for the agent to act so as to minimize the overall cost of performing this repeated task [3], assuming the cost of traversing any edge is 1. Note that we talk only about the cost of action; computation cost is not explicitly considered (though we rule out impracticably expensive methods).

3 The Exploration Algorithms

In this section we describe our utility-based exploration algorithm, as well as two alternative exploration algorithms which we used for comparison purposes.

3.1 EWP: Exploration While Performing tasks

The first algorithm we describe is our utility-based *exploration while performing* (EWP) algorithm. The algorithm attempts incrementally to find shorter paths between A and B while traveling repeatedly between them. At each stage of the exploration, the agent knows the structure of the portion of the environment that it has seen so far. The agent repeatedly travels from A to B and back. In addition, during each trip the agent attempts to find a previously unknown path between some known pair of nodes v_1 and v_2. (In the process of finding such a path, new nodes may also be discovered.) In order to find a new path between v_1 and v_2, the agent uses a heuristic method for graph search (described below) to move through unknown territory from v_1 towards the position of v_2. Such (deterministic) exploration finds a unique *default path* between v_1 and v_2. When this default path between the pair of nodes (v_1, v_2) is not yet known by the agent, the pair is a candidate for exploration; we term such a pair an *exploration edge*. On each trip between A and B, therefore, the agent explores by following a 'path' that contains one or more exploration edges. Since the number of such paths is exponential in the number of exploration edges, we restrict our attention to paths that contain only one exploration edge, terming such paths *exploration paths*.

On each trip, the EWP algorithm evaluates the expected utility of each existing exploration path for the remaining task repetitions, and traverses the one with the highest utility. In our setting, the utility of a path is just the inverse of its cost of traversal [4]. Evaluating the expected utility of an exploration path,

[3] In Section 5 we consider the case of multiple repeated goals, where the goal on each repetition is chosen randomly from a known set of possible goals.

[4] The method does not change greatly, however, if other utility factors, eg, the possibility of refueling, are incorporated.

therefore, requires evaluating the expected cost of traversing an exploration edge [5] (discussed in detail below). We must distinguish between the expected cost of traversing an exploration edge the first time, and the cost of traversing the default path that was discovered on subsequent task repetitions. The first time an exploration edge is traversed, using heuristic search, the agent might need to backtrack or might discover cycles, both of which are not included in the default path.

We term the utility of an exploration path including the single exploration edge e as $U(e)$. We may express the utility $U(e)$ of an exploration path including exploration edge e as the sum of two component utilities:

- $U_0(v_0, e, B)$, the expected utility of a single trip from the agent's current node v_0 to the current target B via e, and
- $U_r(e, R, A, B)$, the expected utility of R future traversals between the target nodes A and B, given that the exploration edge e was already explored.

How U_0 and U_r are computed is described below in Section 3.1.1, where we discuss the default traversal method in detail.

Note that the exploration strategy we have described is *greedy*, in that only one exploration edge is examined on each trip. We improve this greedy method by applying it recursively—after the agent traverses exploration edge (v_1, v_2) and is at v_2, EWP is applied recursively to find the best exploration path between the current node v_2 and B.

The full EWP algorithm is given in Figure 1. The agent evaluates the utility of all outstanding exploration paths (Step 4), and chooses the best path to traverse (Step 6), if a good one exists. If the agent thus arrives at its current target (B), it exchanges A for B and continues with its remaining $R - 1$ traversals. Otherwise, it attempts to explore again from its current position, until it is no longer useful to do so. In that case, the agent uses the best known path to get to its current target (Step 7a).

3.1.1 Discovery and expected cost of default paths

In order to traverse an exploration edge between two known nodes v_1 and v_2, we apply a depth-first search strategy to find v_2 starting from v_1. Following Cucka et al. [9], we use the minimal-angle heuristic to inform the search.

[5] Note that there may be an 'exploration edge' between nodes v_1 and v_2, even if there is no real edge in the environment between them. An exploration edge merely denoted the possibility of exploring the territory between the nodes using the default method, and discovering thereby a new *path* (possibly containing several edges) between them.

Algorithm 1 EWP (A, B, R)

(1) If $R = 0$, terminate;

(2) Let v_0 be the current node of the agent, and U_{known} be the utility of completing the remaining R tasks using only paths in the known environment;

(3) Enumerate the set of exploration edges $\{e^i = (v_1^i, v_2^i)\}$;

(4) Evaluate the expected utility $U(e) = U_0(v_0, e, B) + U_r(e, R-1, A, B)$ of each exploration edge e;

(5) Let $e^* = (v_1^*, v_2^*)$ be the exploration edge with highest expected utility;

(6) If $U(e^*) \geq U_{\text{known}}$, then:

- *(a) Follow the best known path to get to v_1^*;*
- *(b) Move to v_2^* using the default heuristic traversal method;*
- *(c) If $v_2^* = B$, $A \leftrightarrow B$, $R \leftarrow R - 1$;*

(7) Else:

- *(a) Follow the best known path to get to B;*
- *(b) $A \leftrightarrow B$, $R \leftarrow R - 1$;*

(8) Goto 1.

Fig. 1. The EWP on-line exploration algorithm.

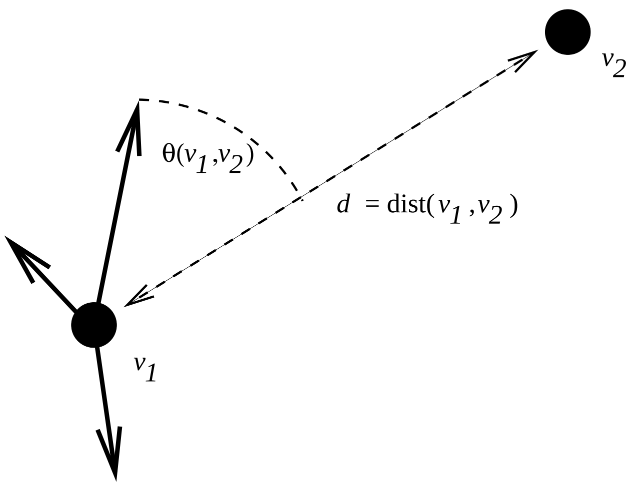

Fig. 2. Distance and angle parameters used in computing $\text{FirstCost}(v_1, v_2)$ and $\text{RestCost}(v_1, v_2)$.

At each node in the search, the agent moves along the incident unexplored edge which most nearly points in the direction of v_2, the search goal. If all edges incident on the current node have previously been traversed, the agent backtracks to the last node it explored. In the worst case, this strategy will explore the entire environment, but in practice this heuristic is often quite efficient [9].

Given the minimal-angle heuristic as a default method for traversing an exploration edge, we now consider how to evaluate the expected cost of such a traversal. Recall that we separately evaluate the cost of the first time the edge is traversed (using heuristic search, including the cost of backtracking and cycles) and the cost of subsequent traversals (using the default path found). In the absence of other, more specific, information, we parameterize the expected cost for exploration between v_1 and v_2 by the distance d between them and the smallest angle θ between an edge from v_1 and the straight line from v_1 to v_2, as depicted in Figure 2. (The smaller θ is, the cheaper the traversal will tend to be.) We term the two expected cost functions $\text{FirstCost}(d, \theta)$ and $\text{RestCost}(d, \theta)$. In order to estimate the repetition utility U_r, we assume that no further exploration will take place (in order to avoid a combinatorial explosion). Since this leads to an overestimate in expected cost, we multiply the raw estimated cost by a heuristic factor $\gamma < 1$. (This only affects the termination condition and not the exploration strategy.)

The utility functions U_0 and U_r are thus defined as follows, where $\text{PathCost}(u, v)$ denotes the cost of the best path known between u and v:

$$
\begin{aligned}
U_0(u_1, e = (v_1, v_2), u_2) = -[\ & \text{PathCost}(u_1, v_1) \\
& + \text{FirstCost}(\text{dist}(v_1, v_2), \theta(v_1, v_2)) \\
& + \text{PathCost}(v_2, u_2)]
\end{aligned}
$$

$$
\begin{aligned}
U_r(e = (v_1, v_2), R, A, B) = -\gamma R[\ & \text{PathCost}(A, v_1) \\
& + \text{RestCost}(\text{dist}(v_1, v_2), \theta(v_1, v_2)) \\
& + \text{PathCost}(v_2, u_2)]
\end{aligned}
$$

We approximate the FirstCost and RestCost functions by assuming that the environment is drawn randomly from a known target class according to a known distribution. The functions are then estimated from simulations performed on an ensemble of environments generated randomly from the target class. A set of such environments are generated and the heuristic search procedure is executed for traversal between each pair of nodes. The average values for FirstCost and RestCost are recorded in a table for each distance and minimal starting angle. This table is then used to evaluate the expected values of FirstCost and RestCost. Our results, described below, show that small errors in identifying the target class do not adversely affect results (see Section 4.4).

3.2 EBP: Exploration Before Performing tasks

As we mentioned above, our main goal is to compare between exploration while performing tasks and exploration before performing tasks. We compared our EWP algorithm with an *exploration before performing* (EBP) algorithm, where the agent studies *all* the edges in the environment using a heuristic backtracking traversal of the graph, and only then moves so as to carry out its tasks. We implemented the algorithm using the minimal-angle search heuristic described above. The agent first explores any unknown edges incident on its current location (in heuristic order for reaching a target node), and backtracks when no such unexplored edges exist. We did not use a more efficient search strategy such as Real-Time A* [22], since such strategies require that the agent be able in one step to know all the children of its current node, which our model disallows.

In early experiments, we found that if EBP is allowed to explore until the entire environment is known, EWP is always considerably more efficient. We therefore introduce a parameter, α, such that the agent stops exploring the environment when the shortest known path between A and B is no longer than α. Once such a path is found, the agent proceeds with its remaining task repetitions using that path. This avoids the problem of diminishing returns of further exploration. Note that although the algorithm's purpose is mainly to explore the environment before performing any tasks, any (incidental) visits to A and B during exploration count as task accomplishments.

3.3 REWP: Randomized Exploration While Performing

We also compared our original algorithm with a simpler algorithm that explores randomly while performing its tasks. This *randomized exploration while performing* (REWP) algorithm is loosely modeled on probabilistic exploration methods used in reinforcement learning (see, eg, [29]). The algorithm generally follows the shortest known path towards its current goal (or uses default search if no path is known), but if the current node has any unknown neighboring edges, with *exploration probability* p_e REWP attempts to traverse one of those unknown edges (chosen randomly). In our experiments below, p_e was set to 0.3, which overall gave the best results. Also, in order that the algorithm not waste too much time exploring fruitlessly, we also introduced here a parameter α, such that REWP stops exploring when the best known path between A and B is shorter than α.

4 Experiments

4.1 The simulation

In order to compare the above exploration algorithms we performed simulations on randomly generated navigation tasks. The basic parameters for the simulations were the number of nodes n in the environments to be considered, the probability p of an edge existing between two nodes, the number of task repetitions R, the heuristic cost multiplier γ (for EWP), and the stopping criterion α (for EBP and REWP). The agent knows R, but does not know n or p. The results reported here are for $\gamma = 0.4$ for random environments and $\gamma = 0.9$ for triangle environments, and $\alpha = 4$, since those values gave the best results overall for the exploration algorithms. Also, EWP assumed that $p = 0.1$ (in order to construct the FirstCost and RestCost tables); results did not seem to be sensitive to the exact value of p. For each experimental trial, we report the average results over an ensemble of 100 randomly generated environments. All of the algorithms were tested on the same ensemble of environments for each trial.

We considered two classes of environments; in both the positions of the nodes were generated uniformly in the unit square. The first type we call *random environments*, where edges are generated between pairs of nodes according to the probability p. Such environments model, for example, tasks in communication networks. Random environments are typically very non-planar, and a small amount of exploration can help a lot, if a good shortcut is found. The other type of environment we considered are *triangle environments*, in which edges are randomly generated based on the Delaunay triangulation [1] of the nodes' positions. These environments model, for example, real-world robot navigation tasks. In triangle environments the potential benefit of exploration may be somewhat less than in random environments, since the likelihood of finding a good shortcut is lower.

4.2 Random environment results

In these experiments, we compared the algorithms' performance on random environments which were generated as follows, given environment size (number of nodes) n and edge probability p:

(1) Begin with the complete graph (V, E) on n nodes, assigning a random weight to each edge.
(2) Find a minimal spanning tree $T = (V, E_T)$ for this weighted graph (to ensure the environment is connected).
(3) Let $E' = E_T$.
(4) Assign each node in V a position chosen uniformly in the unit square.
(5) For each $v_1, v_2 \in V$ such that $(v_1, v_2) \notin E_T$, add (v_1, v_2) to E' with probability $p\frac{n-1}{n}$.
(6) Choose, with probability $\frac{1}{n}$ and $\frac{1}{n-1}$ respectively, two nodes in the graph as goals, A and B.
(7) Output the graph $G = (V, E')$ with target nodes A and B.

	100 Nodes			200 Nodes		
$R\backslash\ \ p$	0.01	0.05	0.1	0.01	0.05	0.1
10	1.08	0.76	0.88	1.28	0.93	0.93
50	0.61	0.44	0.51	0.59	0.47	0.52
100	0.52	0.38	0.45	0.48	0.37	0.42
200	0.48	0.34	0.41	0.41	0.32	0.36

Table 1
Ratio of task performance time of exploration while performing (EWP) over the default path cost, in random environments. We compare performance for different edge probabilities p and different numbers of task repetition R.

In Tables 1 and 2, we compare the overall efficiency of EWP for repeated tasks versus using the default traversal method. We see that except with very few repeats, efficiency is always improved by using on-line exploration. Furthermore, we note that as the number of task repetitions increases, the usefulness of exploration increases. This is because the cost of an exploratory action can be amortized over a larger number of future tasks. The effectiveness of exploration is highest in environments of intermediate 'edge density' ($p = 0.05$), because while (i) a dense environment contains more shortcuts for exploration to find than a sparse one, (ii) the direct (default) path in a dense environment is more likely to be short than in a sparse environment. These two influences are balanced in environments of intermediate density.

In Tables 3 and 4 we compare the performance of the three exploration algorithms (EWP, EBP, and REWP) against each other for random environments. We tested our results statistically using analysis of paired data [14, Section 9.3] to compare the algorithms' mean efficiencies. To a 0.05 confidence level, EWP is more efficient than EBP and REWP, while REWP is more efficient than EBP.

More specifically, for the cases considered, on-line exploration (EWP and REWP) often performs much better than off-line exploration (EBP), and is never noticably worse. In addition, the utility-based focus of EWP is of great advantage, as EWP achieves up to a 38% efficiency advantage over REWP. The additional focusing of exploration provided by the utility-based formulation is also seen in that the advantage of EWP over the other algorithms increases with the density of the environment, since EWP is not distracted by additional irrelevant edges. The usefulness of on-line exploration increases with increasing size of the environment for the same reason. As expected, the usefulness of on-line versus off-line exploration decreases with an increasing number of task repetitions, since the cost of exploration can be amortized over more repetitions of the task.

$R = 100$			
$n\backslash\ \ p$	0.01	0.05	0.1
10	0.90	0.91	0.84
20	0.89	0.72	0.69
30	0.79	0.59	0.61
40	0.68	0.52	0.52
50	0.60	0.45	0.51
60	0.65	0.51	0.51
70	0.58	0.45	0.54
80	0.57	0.47	0.53
90	0.55	0.46	0.46
100	0.52	0.38	0.45
200	0.48	0.37	0.42

Table 2
Ratio of task performance time of exploration while performing (EWP) over the default path cost, in random environments. We compare performance for different edge probabilities p and different environment sizes n.

$R = 100$	EWP/EBP			EWP/REWP			REWP/EBP		
$n\backslash\ \ p$	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10
10	0.96	0.95	0.94	0.97	0.97	0.94	0.99	0.98	1.00
20	1.00	0.92	0.85	1.01	0.94	0.89	0.99	0.98	0.96
30	1.01	0.88	0.79	1.04	0.91	0.84	0.97	0.97	0.94
40	0.94	0.85	0.77	0.96	0.90	0.80	0.98	0.94	0.96
50	0.95	0.82	0.84	0.94	0.88	0.84	1.01	0.93	1.00
60	0.93	0.83	0.83	1.00	0.86	0.81	0.94	0.96	1.02
70	0.92	0.81	0.82	0.99	0.85	0.85	0.93	0.94	0.96
80	0.89	0.82	0.80	0.99	0.88	0.85	0.90	0.93	0.94
90	0.87	0.77	0.76	0.97	0.90	0.76	0.89	0.86	1.00
100	0.82	0.70	0.65	0.88	0.80	0.68	0.93	0.88	0.96
200	0.67	0.64	0.66	0.91	0.65	0.63	0.73	1.00	1.05

Table 3
Task efficiency ratios between exploration algorithms for random environments of different sizes (n), for $R = 100$ task repetitions.

100 Nodes	EWP/EBP			EWP/REWP			REWP/EBP		
$R\backslash \quad p$	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10
10	0.41	0.42	0.47	0.75	0.53	0.59	0.55	0.78	0.79
50	0.71	0.65	0.62	0.86	0.75	0.66	0.82	0.86	0.94
100	0.82	0.70	0.65	0.88	0.80	0.68	0.93	0.88	0.96
200	0.90	0.71	0.66	0.97	0.76	0.71	0.92	0.94	0.94

200 Nodes	EWP/EBP			EWP/REWP			REWP/EBP		
$R\backslash \quad p$	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10
10	0.30	0.37	0.40	0.57	0.34	0.37	0.53	1.09	1.07
50	0.53	0.60	0.62	0.84	0.61	0.58	0.63	0.99	1.07
100	0.67	0.64	0.66	0.91	0.65	0.63	0.73	1.00	1.05
200	0.78	0.69	0.67	0.94	0.64	0.62	0.84	1.07	1.09

Table 4
Efficiency ratios comparing EWP, EBP, and REWP for varying numbers of task repetitions R and edge probabilities p, in random environments.

4.3 Triangle environment results

We also compared the algorithms' performance on triangle environments. As noted above, we examined performance on triangle environments as well as random environments, since triangle environments more closely approximate the situation for robotic navigation problems. Furthermore, we expect exploration in general to be less effective for triangle environments, due to the lack of shortcuts; therefore we tested if our positive results for exploration while performing hold for the triangle case as well. Triangle environments were generated as follows, given environment size (number of nodes) n and edge probability p.

Triangle environments with an overall edge probability of p were generated as subsets of Delaunay triangulations via the following procedure:

(1) Generate n points randomly in a unit square (with a uniform distribution).
(2) Compute the Delaunay triangulation $G = (V, E)$ of the points (using the `qhull` software package [1]).
(3) Assign each edge a random weight.
(4) Find a minimal spanning tree $T = (V, E_T)$ for this weighted graph.
(5) Let $E' = E_T$.

$R = 100$	EWP/Def.			EWP/EBP			EWP/REWP			REWP/EBP		
$n\backslash\ \ p$	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10
10	0.97	0.98	0.96	1.03	1.03	0.98	1.03	1.05	0.99	1.00	0.98	0.99
20	0.99	0.86	0.83	1.05	1.04	0.99	1.07	1.03	1.01	0.98	1.01	0.98
30	0.95	0.77	0.85	1.06	1.02	0.98	1.07	1.04	1.00	0.99	0.98	0.97
40	0.88	0.77	0.88	1.04	0.94	0.92	1.07	0.99	0.99	0.98	0.95	0.92
50	0.79	0.76	0.88	1.02	0.88	0.88	1.04	0.97	0.96	0.99	0.91	0.92
60	0.78	0.83	0.85	0.98	0.89	0.86	1.00	0.99	0.97	0.98	0.90	0.88
70	0.71	0.85	0.88	0.92	0.79	0.79	0.93	0.96	0.96	0.99	0.83	0.82
80	0.72	0.77	0.77	0.91	0.75	0.75	0.95	0.93	0.93	0.96	0.81	0.81
90	0.72	0.81	0.81	0.88	0.71	0.70	0.96	0.92	0.92	0.91	0.76	0.76
100	0.75	0.85	N/A	0.89	0.67	N/A	1.00	0.92	N/A	0.89	0.73	N/A
200	0.65	0.76	N/A	0.64	0.52	N/A	0.95	0.92	N/A	0.67	0.57	N/A

Table 5
Comparison of efficiency ratios of algorithms EWP, EBP, REWP, and the default method for triangle environments of varying size (n), for $R = 100$ task repetitions.

(6) For each edge $e \in E - E_T$, add e to E' with probability

$$q = p\frac{n(n-2)}{2(|E| - n + 1)} \quad ,$$

where p is the input edge probability, n is the number of nodes in the graph and $|E|$ is the number of edges in the triangulation graph.

(7) Choose, with probability $\frac{1}{n}$ and $\frac{1}{n-1}$ respectively, two nodes in the graph as targets, A and B.

(8) Output the environment $G = (V, E')$, and goals A and B.

The probability q for adding an edge in the Delaunay triangulation to G is computed so that the expected fraction of all possible edges that are edges in G is p. In graphs of 100 nodes or more, the Delaunay triangulation had fewer than $0.01n(n-1)$ edges, so there are no graphs for $p = 0.10$ and $n \geq 100$.

Tables 5 and 6 give the results of comparing the various algorithms for triangle environments. We first see that EWP is still significantly preferred to the default path, although its benefit is decreased compared to random environments. This is due to the lower incidence of shortcuts in the planar triangle environments. As in random environments, the usefulness of exploration increases with the size of the environment and the number of task repetitions. On-line exploration (EWP) also gave efficiency improvements over off-line exploration (EBP) similar to those in the random environment case, showing

100 Nodes	EWP/Def.		EWP/EBP		EWP/REWP		REWP/EBP	
$R \backslash \quad p$	0.01	0.05	0.01	0.05	0.01	0.05	0.01	0.05
10	1.05	1.07	0.36	0.18	0.73	0.42	0.49	0.41
50	0.79	0.89	0.73	0.49	0.95	0.80	0.77	0.62
100	0.75	0.85	0.89	0.67	1.00	0.92	0.89	0.73
200	0.74	0.83	1.01	0.83	1.03	1.00	0.99	0.84

200 Nodes	EWP/Def.		EWP/EBP		EWP/REWP		REWP/EBP	
$R \backslash \quad p$	0.01	0.05	0.01	0.05	0.01	0.05	0.01	0.05
10	0.97	1.15	0.17	0.12	0.55	0.43	0.31	0.28
50	0.70	0.85	0.46	0.36	0.87	0.82	0.52	0.44
100	0.65	0.76	0.64	0.52	0.95	0.92	0.67	0.57
200	0.63	0.71	0.82	0.70	1.00	0.99	0.81	0.71

Table 6
Comparison of efficiency ratios of algorithms EWP, EBP, REWP, and the default method for triangle environments of varying size (n), for $R = 100$ task repetitions.

that the focus on task-related exploration is still significant. However, the utility-based focus of EWP was less important in triangle environments, as seen in the efficiency ratio of EWP over REWP. For over 100 task repetitions (and all environment sizes), the two algorithms perform virtually identically, showing that with sufficient amortization of exploration cost, randomized on-line exploration is as good as utility-based exploration. This is because random exploration in a triangle environment is unlikely to move the agent far from the currently known best path to the target, so that the agent tends to explore those portions of the environment more likely to be relevant to the task. In random environments, on the other hand, a random step in the graph may move the agent far from the portion of the environment between the targets, and so the agent wastes more time exploring irrelevant portions of the environment.

In triangle environments, the usefulness of exploration decreases with increasing density, since the shortest path will tend to be more direct, the more edges there are. The usefulness of on-line exploration versus off-line exploration increases, however, with increasing density, since the off-line algorithm will waste more time exploring irrelevant portions of the environment.

		EWP/Low			EWP/Avg			EWP/High		
n	$R\backslash\ p$	0.01	0.05	0.10	0.01	0.05	0.10	0.01	0.05	0.10
100	50	0.70	0.72	0.64	0.60	0.58	0.54	0.88	0.98	0.94
100	100	0.63	0.64	0.58	0.54	0.51	0.49	0.82	0.92	0.89
200	50	0.68	0.75	0.77	0.62	0.61	0.57	0.87	0.97	1.04
200	100	0.60	0.61	0.64	0.54	0.59	0.47	0.81	0.87	0.91

Table 7
Comparison of the EWP algorithms with the Low, Avg, and High variants for random environments (see text for full explanation).

		EWP/Low		EWP/Avg		EWP/High	
n	R	0.01	0.05	0.01	0.05	0.01	0.05
100	50	0.76	0.88	0.76	0.88	0.82	0.92
100	100	0.74	0.85	0.74	0.85	0.80	0.91
200	50	0.68	0.89	0.68	0.83	0.82	0.99
200	100	0.64	0.80	0.64	0.75	0.78	0.93

Table 8
Comparison of the EWP algorithms with the Low, Avg, and High variants for triangle environments (see text for full explanation).

4.4 Sensitivity of EWP

We evaluated the sensitivity of EWP to its heuristic (the tables of FirstCost and RestCost) in two ways. First, as noted above, in all of our experiments we used tables of FirstCost and RestCost based on $p = 0.1$ and $n = 100$. We found EWP to be significantly useful in all classes of environments, and hence detailed tuning of these parameters does not appear to be crucial.

Second, we evaluated the need for a detailed evaluation of expected utility by comparing the EWP algorithm using the FirstCost and RestCost tables against a version of the same algorithm which used constant values for the FirstCost and RestCost functions. We evaluated three variations, one using the lowest value in the table (Low), one using the highest (High), and one using the average of all the values (Avg). The results are summarized in Tables 7 and 8. These results show that using the tables of values gives significant efficiency improvement over using a fixed value, indicating the usefulness of utility evaluation in exploration. We also see that the best of the constant-value algorithms is High, reflecting the earlier termination effected by overestimating (rather than underestimating) the expected path cost.

5 Multiple Repeated Tasks

In the previous sections, we have described and evaluated algorithms for combining exploration and task achievement in a partially-unknown environment, where the agent is to perform a single task repeatedly. In this section, we report on some results for the more common case where the agent may have to perform a number of different tasks in sequence. We model this case by assigning each node a probability of being chosen as a navigational target on each repetition. Thus, the agent must travel to R target nodes, where the identity of the target node for task number i is chosen randomly after the agent reaches target $i - 1$. We assume that the task probability distribution is static and is known by the agent, though the agent does not know what its specific future goals will be. In this work, we assume a uniform probability distribution over a subset $S \subset V$ of possible target nodes. The agent starts out only knowing the set of nodes S (with their positions).

We generalized the exploration algorithms described above to the case of multiple goals in a straightforward manner. For the exploration before performing (EBP) and randomized exploration while performing (REWP) algorithms, the only change necessary is in the termination criterion, where exploration is terminated if (i) all target nodes have been visited and (ii) the average shortest path length between pairs of possible goals is less than the threshold α. The utility-based exploration while performing (EWP) algorithm is adjusted by computing the expectation of future utility of possibly discovering a new edge over all possible sequences of goals. This is done by computing the average improvement in shortest expected path length for each exploration path, over all pairs of possible goals.

In general, we expect the efficiency of on-line exploration versus off-line exploration to decrease as the number of different goals increases, since the focus that on-line exploration provides will be more diffuse. Our results are presented in Tables 9 and 10.

Surprisingly, we find that EWP often performs much better than EBP, even for large numbers of goals. In random environments (Table 9), the advantage of EWP over EBP is noticeable even for large numbers of targets and many task repetitions, particularly in dense environments. This result shows that in dense random environments, there is significant overlap between the good paths for different goals, and so the task-based focus of on-line exploration can still be used to some advantage. In addition, in random environments the utility-based focus of EWP gives it an advantage over REWP, just as in the two-target case. As in that case, the advantage of EWP increases for fewer task repetitions.

$p = 0.01$	EWP/Def.				EWP/EBP				EWP/REWP			
$\|S\|\backslash \quad R$	100	500	1000	2000	100	500	1000	2000	100	500	1000	2000
2	0.51	0.44	0.44	0.43	0.87	0.98	1.00	1.01	0.98	0.98	0.98	1.00
5	0.77	0.68	0.66	0.66	0.85	0.95	0.96	0.97	0.96	0.98	0.98	0.98
10	0.93	0.85	0.84	0.83	0.91	0.97	0.98	0.98	1.01	1.00	0.99	0.99
20	0.95	0.94	0.93	0.93	0.96	0.98	0.99	0.99	1.01	1.00	0.99	0.99
30	0.94	0.94	0.95	0.95	0.98	0.99	0.99	0.99	1.03	1.00	1.00	1.00
$p = 0.05$	EWP/Def.				EWP/EBP				EWP/REWP			
$\|S\|\backslash \quad R$	100	500	1000	2000	100	500	1000	2000	100	500	1000	2000
2	0.46	0.42	0.42	0.42	0.83	0.92	0.93	0.94	0.83	0.89	0.90	0.90
5	0.47	0.50	0.48	0.48	0.68	0.78	0.79	0.80	0.80	0.80	0.79	0.82
10	0.79	0.66	0.64	0.62	0.67	0.82	0.85	0.86	0.87	0.85	0.85	0.85
20	0.90	0.80	0.77	0.75	0.68	0.85	0.88	0.90	0.91	0.93	0.93	0.91
30	0.91	0.84	0.82	0.80	0.67	0.87	0.91	0.93	0.92	0.96	0.96	0.94
$p = 0.10$	EWP/Def.				EWP/EBP				EWP/REWP			
$\|S\|\backslash \quad R$	100	500	1000	2000	100	500	1000	2000	100	500	1000	2000
2	0.51	0.49	0.49	0.49	0.89	0.94	0.95	0.95	0.81	0.91	0.90	0.93
5	0.52	0.45	0.44	0.44	0.58	0.74	0.78	0.79	0.71	0.76	0.75	0.75
10	0.69	0.56	0.36	0.54	0.53	0.78	0.57	0.89	0.77	0.84	0.56	0.84
20	0.82	0.71	0.69	0.67	0.56	0.84	0.91	0.96	0.81	0.92	0.94	0.94
30	0.88	0.79	0.76	0.74	0.53	0.84	0.93	0.99	0.82	0.96	0.98	0.98

Table 9
Efficiency ratios of exploration algorithms for multiple targets in random environments of size $n = 100$, with varied R and p.

In triangle environments (Table 10), on-line exploration is preferred, even for large numbers of targets, in cases with relatively low numbers of repetitions. EWP and REWP perform nearly identically, however, so, as in the two-target case, for more than 100 task repetitions explicit evaluation of utility is not indicated for triangle environments.

Finally, we note that in all cases, EWP performs better than the default path, even with many goals (although the advantage decreases with the number of goals). As expected, the improvement increases with the number of repetitions, as more of the environment can be learned.

$p = 0.01$	EWP/Def.				EWP/EBP				EWP/REWP			
$\|S\|\backslash\ \ R$	100	500	1000	2000	100	500	1000	2000	100	500	1000	2000
2	0.71	0.62	0.62	0.61	0.92	0.99	1.00	1.00	1.03	0.99	0.98	0.98
5	0.79	0.69	0.67	0.67	0.92	0.99	1.00	1.00	1.02	0.99	0.99	0.98
10	0.89	0.82	0.81	0.80	0.96	0.99	1.00	1.00	1.03	1.00	1.00	1.00
20	0.92	0.91	0.91	0.91	0.98	0.99	1.00	1.00	1.02	1.00	1.00	1.00
30	0.92	0.94	0.94	0.95	1.03	1.01	1.00	1.00	1.02	1.00	1.00	1.00
$p = 0.05$	EWP/Def.				EWP/EBP				EWP/REWP			
$\|S\|\backslash\ \ R$	100	500	1000	2000	100	500	1000	2000	100	500	1000	2000
2	0.96	0.75	0.72	0.71	0.82	0.94	0.97	0.99	1.07	1.01	1.00	1.01
5	0.88	0.61	0.57	0.55	0.75	0.90	0.93	0.95	1.06	1.00	0.98	0.98
10	0.93	0.65	0.61	0.59	0.75	0.91	0.95	0.97	1.03	1.01	1.00	1.00
20	0.97	0.76	0.72	0.70	0.76	0.91	0.95	0.97	1.02	1.01	1.00	1.00
30	0.99	0.82	0.80	0.78	0.77	0.92	0.95	0.97	1.04	1.01	1.01	1.00

Table 10

Efficiency ratios of exploration algorithms for multiple targets in triangle environments of size $n = 100$, with varied R and p.

6 Discussion

The EWP algorithm presented above uses a greedy estimate of exploration utility in order to explore a partially-known environment on-line during performance of repeated tasks. Our results for a number of different types of graphs show on-line exploration to be generally superior to a good off-line exploration algorithm (EBP). We further have shown that EWP usually gives better performance than a randomized on-line exploration algorithm (REWP), demonstrating the importance of explicitly considering the expected utility of exploration. Utility-based on-line exploration only explores parts of the environment that are expected to help the agent with its given tasks, avoiding exploring portions of the environment that are irrelevant to the agent. One surprising result is that EWP also gives significant efficiency improvements in some cases with more than two repeated goals, because even in such cases most of the environment need not be explored in order to perform the required tasks efficiently. We conclude that in some cases a utility-based on-line exploration strategy can achieve a significant improvement in agent performance.

References

[1] C. B. Barber and H. Huhdanpaa. Qhull software package, 1995. `http://www.geom.umn.edu/software/qhull`.

[2] K. Basye, T. Dean, and J. S. Vitter. Coping with uncertainty in map learning. Technical Report CS-89-27, Brown University Department of Computer Science, June 1989.

[3] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[4] D. A. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments.* Chapman and Hall, London, UK, 1985.

[5] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[6] M. Betke, R. L. Rivest, and M. Singh. Piecemeal learning of an unknown environment. *Machine Learning*, 18(2/3):231–254, 1995.

[7] A. Blum and P. Chalasani. An on-line algorithm for improving performance in navigation. In *Proc. Symp. Foundations of Computer Science*, pages 2–11, 1993.

[8] F. Chimura and M. Tokoro. The trailblazer search: A new method for searching and capturing moving targets. In *Proc. National Conference on Artificial Intelligence*, pages 1347–1352, 1994.

[9] P. Cucka, N. S. Netanyahu, and A. Rosenfeld. Learning in navigation: Goal finding in graphs. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(5), 1996.

[10] T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, and O. Maron. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning*, 18(1):81–108, 1995.

[11] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1–2):35–74, 1995.

[12] X. Deng, T. Kameda, and C. Papadimitriou. How to learn in an unknown environment. In *Proc. of the 32nd Symposium on the Foundations of Comp. Sci.*, pages 298–303. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[13] X. Deng and C. H. Papadimitriou. Exploring an unkhown graph. In IEEE, editor, *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 355–361. IEEE Computer Society Press, Oct. 1990.

[14] J. L. Devore. *Probability and Statistics for Engineering and Sciences.* Brooks/Cole Publishing Company, Pacific Grove, California, 1991.

[15] O. Etzioni. Embedding decision-analytic control in a learning architecture. *Artificial Intelligence*, 49:129–159, 1991.

[16] P. Haddawy, A. Doan, and R. Goodwin. Efficient decision-theoretic planning: Techniques and empirical analysis. In *Proc. Conference on Uncertainty in Artificial Intelligence*, pages 229–236, 1995.

[17] T. Ishida and R. Korf. A moving target search: A real-time search for changing goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):609–619, 1995.

[18] T. Ishida and M. Shimbo. Improving the learning efficiencies of realtime search. In *Proc. National Conference on Artificial Intelligence*, pages 305–310, Portland, OR, 1996.

[19] L. P. Kaelbling. *Learning in Embedded Systems.* The MIT Press, Cambridge, MA, 1993.

[20] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[21] G. I. Karakoulas. Probabilistic exploration in planning while learning. In P. Besnard and S. Hanks, editors, *Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pages 352–361, 1995.

[22] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2–3):189–211, 1990.

[23] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.

[24] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[25] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, 1994.

[26] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences (extended abstract). In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 411–420, Seattle, Washington, 1989.

[27] Y. Smirnov, S. Koenig, M. M. Veloso, and R. G. Simmons. Efficient goal-directed exploration. In *Proc. National Conference on Artificial Intelligence*, pages 292–297, Portland, OR, 1996.

[28] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. Seventh Int'l Conf. on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.

[29] S. Thrun. The role of exploration in learning control. In *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Florence, Kentucky 41022, 1992. Van Nostrand Reinhold.