Interleaved vs. A Priori Exploration for Repeated Navigation in a Partially-Known Graph*

Shlomo Argamon-Engelson[1] Sarit Kraus[1,2] Sigalit Sina[1]

[1] Dept. Mathematics and Computer Science
Bar-Ilan University
52900 Ramat Gan, Israel
Tel: +972-3-531-8407
Email: {argamon,sarit,sina}@cs.biu.ac.il

[2]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 software agent operating on the World-Wide Web may need to learn which sites on the net are most useful to it, and the most efficient routes to those sites. We compare exploration strategies 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 partially known (deterministic) graph. This paper describes a new *utility-based* exploration algorithm for repeated tasks which interleaves exploration with task performance. The method takes into account both the costs and the potential benefits (for future task repetitions) of different exploratory actions. Exploration is performed in a greedy fashion, with the locally optimal exploratory action performed during each task repetition. We experimentally evaluated our utility-based interleaved exploration algorithm against a heuristic search algorithm for exploration before task performance (a priori exploration) as well as a randomized interleaved exploration algorithm. We found that for a single repeated task, utility-based interleaved 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 (from a known subset of possible tasks), task to perform each time. Here too, we found that utility-based interleaved exploration is clearly in most cases.

Keywords: Exploration vs. exploitation, navigation, random graphs, expected utility

*This research was supported in part by the National Science Foundation grant number IRI-9311988 and Israeli Science Ministry grant No. 6288. The first author was 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 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 *a priori*, before performing any tasks, or it may be *interleaved* with task performance. In interleaved exploration, exploration and exploitation must be traded off in order to maximize the overall efficiency of the agent's behavior.

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 software agent operating on the World-Wide Web may need to learn which sites on the net are most useful to it, and the most efficient routes to those sites. Another example is a mobile robot which needs to learn a map of its environment so that it can navigate effectively from place to place. We model such situations by viewing the world as a graph whose nodes represent states, and whose arcs represent actions which move the agent from one state to another (we describe our model more precisely in Sections 2 and 4). By modeling an environment as a state graph, any assigned task becomes a problem of effectively 'navigating' between states in the graph. We assume that the world graph is fully connected and can be entirely represented by the agent.

In this paper, we compare interleaved and a priori 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 begins with the ability to perform its given task(s), but not necessarily in an optimal way. For example, the agent may have access to a teacher which, for a price, will give a (suboptimal) plan for a given task. In the work described here, we assume that the agent begins knowing some navigation plan for every possibly task, given in the form of a spanning tree. The idea is that with interleaved exploration, performance of the task becomes more efficient over time, as the agent learns better ways to perform it [9]. As the number of repetitions gets very large, however, 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 many realistic scenarios, this is not the case, since the number of repetitions of a particular task is bounded. In such cases, it is more effective to learn during task performance, and to 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 [16, 14, 17]. We describe in this paper a new *utility-based* interleaved exploration algorithm for a partially known graph. In our approach, the agent estimates the expected utility of possible exploratory actions, based on a probabilistic model of the structure of the actual world graph. This evaluation enables the agent to decide whether and how to explore. Utility is evaluated relative to the agent's future tasks and the cost of exploring, taking into account the extra cost of exploring versus 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 of exploration may be higher than the expected performance 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 (compare Etzioni's [16] use of a greedy marginal utility heuristic).

We experimentally evaluated utility-based interleaved exploration on a variety of randomly generated graphs, and compared the performance of our greedy utility-based interleaved algorithm against that of a heuristic search algorithm for a priori exploration. Our results show that utility-based interleaved exploration is nearly always preferable to a priori exploration. Furthermore, we investigated the contribution of the utility-based formulation, by comparing it to a randomized approach to interleaved exploration (similar to methods commonly used in reinforcement learning [30, 22]). We found that for a single repeated task in our model, utility-based interleaved 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 (from a known subset of possible tasks), task to perform each time. Here too, we found that utility-based interleaved exploration performs better than either a priori or randomized interleaved exploration.

2 Graph-Based Exploration Problems

In this paper we evaluate a utility-based interleaved exploration strategy for a particular class of graph navigation problems. Our longer term research objective is to investigate a particular class of utility-based interleaved exploration strategies in a variety of graph navigation scenarios, under different assumptions as to the underlying structure of the graph, the capabilities of the navigating agent and the agent's state of knowledge.

The class of strategies we seek to investigate achieve each repetition of the assigned task via an *exploratory path*, a path which includes the exploration of some unknown portion of the graph. Such utility-based strategies are variants of the following generic algorithm for interleaved exploration:

1. Enumerate a set of possible exploratory paths to achieve the current task;
2. Evaluate, for each such path, the expected utility for the remaining task repetitions if the path is taken;
3. If the best exploratory path has higher expected utility than the best path currently known, use it;

4. Else, use the best path currently known.

The use of such a strategy requires the ability to enumerate a set of exploratory paths as well as a method for evaluating the expected utility of such paths. This utility must take into account both the cost of exploration on the current task run and the potential efficiency improvements for future task runs. In the remainder of this section, we discuss several different graph navigation models, including the one addressed specifically in this paper, as well as describing briefly how our framework for utility-based interleaved exploration can be applied in each model.

The graph exploration scenario examined in this paper is inspired by the problem of an agent finding efficient routes in a communications network, eg, the Internet. In this scenario, the agent knows about all of the nodes in the graph, all of which can be uniquely identified. The agent's task is to repeatedly move back and forth some given number of times between two particular nodes in the graph, traversing as few edges in the process as possible. The possible actions that can be taken at a node consist of attempting to move directly to another node; if the corresponding edge exists, the action succeeds and the agent moves to the specified destination, otherwise the agent fails and remains where it is. As an abstract approximation to the situation where the agent can ask a teacher for a default path to its goal, we also assume that the agent begins knowing some path between every pair of nodes. In our greedy interleaved exploration algorithm for this scenario, the set of exploratory paths are those potential paths including one unknown 'edge'. The expected utility of each such path is evaluated based on combining the cost and (known) probability of a failure to traverse the edge with the probability and magnitude of the reduction in total travel cost for future repetitions if the edge exists.

A more complex model, which we are currently investigating in other work [2, 31], is given by relaxing some of the assumptions from the last scenario. First, the agent does not know about all of the nodes in the environment in advance (and so must discover them during exploration). The possible actions the agent can take from a particular node, therefore, are defined be the set of edges incident on the node. That is, the agent does not attempt to go directly to some known node, but rather traverses an actual edge whose destination is unknown. In order to evaluate the utility of potential exploratory actions, some more information about the structure of the environment is needed, so we assume that the agent knows the geometric positions of nodes that it has visited, as well as the directions of the arcs incident on its current node. Exploration in this model involves the attempt to search for a new path between two known nodes, so that an exploration path may be defined as a path which includes traversal of an unknown area between two known nodes. For example, this scenario may be used to model robotic navigation, particularly repetitive delivery tasks. An approximate evaluation of exploration utility may be performed in this model based on the relative positions of source and goal nodes and the directions of the possible arcs to traverse.

Both of the aforementioned scenarios can be generalized further in a number of ways. Actions may be non-deterministic, traversing different arcs from a node with different probabilities. Nodes may not have unique identifiers, such that the agent must somehow infer from other information which node it is currently located at. A related issue is geometric noise in our second model above, where the agent does not know its precise geometric location. The question of prior knowledge is important; in different real-life situations, many kinds of 'taught' information may be available, for some cost. Also, long-range sensing may also improve exploration efficacy, by giving the agent an approximate idea of the structure of the graph in its vicinity. Finally, the properties of the agent's actions/sensing may change regularly over time, for example where the cost of edges changes, or where certain paths become blocked or unblocked with some regularity. It would be useful to discover and make use of such regularities in navigation and exploration.

3 Related Work

Graph-based world models have long been used in theoretical work on a priori map-learning, such as [27, 15, 25, 1]. These models have also been extended to include sensor and effector noise [4, 13]. The main problem that is addressed in such work is that of figuring out how to distinguish different nodes that look the same to the agent. This is typically done by discovering 'distinguishing sequences' which allow the robot to figure out its location based on the results of sequences of actions.

A task situated between a priori and interleaved exploration (not necessarily in a graph-based framework), is piecemeal exploration [8], in which the robot must return to its 'home' every so often. The techniques used are similar to those in a priori exploration, constrained by the necessity of returning home when needed.

State-space learning during task performance has been addressed by so-called 'real-time' search algorithms [23, 11, 18]. 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 [28] and by not requiring the algorithm to find an optimal policy [19]. In more specific problem settings, more informed heuristics may be applied, for example Cucka et al. [12] use information about the geometric direction of the goal to heuristically improve a depth-first search exploration process.

Methods for solving Markov decision processes (MDPs) also involve an interleaved tradeoff between exploration and exploitation [26]. In these problems, the environment is modeled as a probabilistic finite-state machine where the agent receives rewards for being in particular states, and the transition to a new state occurs with some probability depending on what action the agent takes. 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 [5, 7], as well as its simpler variant the k-armed bandit problem [6], but these solutions are of exponential complexity [21]. Various faster approximate strategies have also been proposed for this problem [20, 24, 29] and have been shown to be useful.

The specific graph navigation problem examined in this paper may be cast as a MDP problem with a changing reward function. As such, it is probably not difficult to adapt the iterative update or randomized exploration methods for MDPs to the graph exploration problem addressed in this paper. In particular, variations on randomized strategies [21] are applicable to our problem, and we compare such a strategy against our utility-based method below. We did not examine variations on iterative update techniques, however, since we wish to investigate a class of methods that are generally applicable to a variety of graph navigation problems (as discussed in the previous section), some of which do not fit easily into the MDP framework.

Our approach of incremental utility-based exploration may also be compared to the anytime algorithm of Dean et al. [14] 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.

4 Problem Definition

We consider the following problem of repeated navigation in a partially known graph. The world is defined to be an undirected graph $G = (V, E)$ where V is a set of N nodes and E is the set of edges of the graph. The agent knows all the nodes of the graph, but is not familiar with all the edges of the graph. However, the agent is given a spanning tree $T = (V, E_T)$ of the graph and, for any $v_1, v_2 \in V$, if $(v_1, v_2) \notin E_T$ the probability that $(v_1, v_2) \in E$ is some probability p, where $0 < p < 1$. We assume in this work that the agent knows this probability[1].

In order to test if the edge $(v_1, v_2) \in E$, the agent must be located at one of the nodes v_1 or v_2, and attempt to move to the other node. If $(v_1, v_2) \in E$ then the agent succeeds in reaching the other node and pays a fixed[2] cost of 1. However, if $(v_1, v_2) \notin E$ than the action fails and the agent pays a (known) failure cost, denoted by c. There is no other way for the agent to find out whether an edge not in T exists.

The agent is given the following task to perform. There are two distinguished nodes in V, A and B, and the agent is to go from A to B and back some given number of times, denoted by R. We would like the agent to act so as to minimize the overall cost of performing this single repeated task[3]. Note that we talk only about the cost of action; computation cost is not explicitly considered (though we rule out impractically expensive methods).

[1] In our experiments (Section 6 below), we also tested the sensitivity of our results to errors in the agent's knowledge of p. It appears that knowing the precise value of p is not crucial.

[2] This simple model can be extended by allowing different costs for different actions, or by making the cost probabilistic. Our methods can easily be extended, provided we assume that the agent knows the expected cost of a contemplated action.

[3] In Section 7 we consider the case of multiple repeated goals, where the goal on each repetition is chosen randomly from a known set of possible goals.

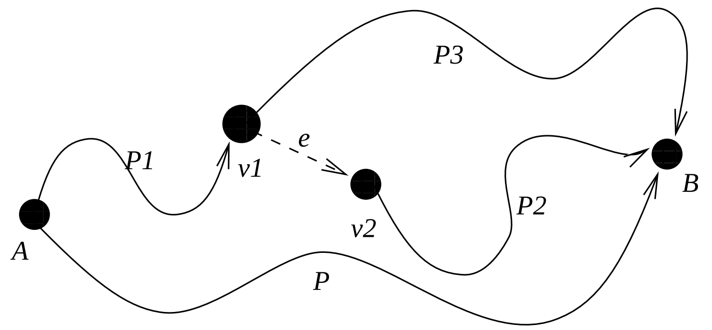


Figure 1: A possible exploratory path, testing the edge (v_1, v_2).

5 The Exploration Algorithms

5.1 LWP: Learning While Performing tasks

The first algorithm we describe is our utility-based *learning while performing* (LWP) algorithm. The algorithm tries to incrementally find shorter paths from A to B in the graph while traveling repeatedly between them. At each stage of the exploration, the agent knows about a set of existing edges and a set of non-existent edges (i.e, that there is no edge between certain pairs of nodes). During each trip, the agent obtains information about one unknown edge that could reduce the travel distance from A to B. Consider the situation depicted in Figure 1, in which the currently-known shortest path between A and B is P. The agent now considers the set of possible paths between A and B that are (i) shorter than P, and (ii) include just *one* unknown edge $e = (v_1, v_2)$. In the figure, such a path is depicted as the path $P_1 \cdot e \cdot P_2$. We term such paths *exploratory paths*. The robot then attempts to travel along the exploratory path with the lowest expected cost (described below). If the agent reaches the destination by following this path, then it has found a shorter path between A and B which it can use in the future. If the agent fails to traverse the path because the unknown edge e does not exist, the agent takes the shortest path from the failure point to its destination (P_3 in the figure), and records that the edge does not exist. The search for shorter paths terminates when the expected cost for checking a new path is higher than using the best known path.

In order to compare the value of different exploratory paths, we wish to evaluate the overall utility of taking each such path, including its effect on future tasks. That is, we wish to measure the expected cost of completing the assigned R repetitions of the task, given that a particular exploratory path is used to navigate for the current repetition. This expectation is measured over the probability p that the path's unknown edge actually exists.

Consider an exploratory path including the unknown edge $e = (v_1, v_2)$. Let P, P_1, P_2, P_3 respectively be the shortest paths in the graph, including only known edges, between A and B, between A and v_1, between v_2 and B, and between v_1 and B respectively (as in Figure 1). The current default cost of a task repetition is $L_o = |P|$, the length of the currently known best path. The cost of exploring e is $L_n = |P_1| + |P_2| + 1$ if e exists, and $L_f = |P_1| + |P_3| + c$ otherwise (where

c is the cost of failing to traverse e). Hence, we wish to compute the expected cost:

$$\begin{array}{rl} p & [L_n + (\text{Cost of the remaining } R-1 \text{ repetitions if } e \text{ exists})]+ \\ (1-p) & [L_f + (\text{Cost of the remaining } R-1 \text{ repetitions if } e \text{ does not exist})] \end{array}$$

In order to accurately evaluate the cost of the remaining repetitions, however, we would have to consider all possible exploratory paths in the future, which leads to exponential complexity. In our early tests, we tried using a lookahead of 2, taking into account the possibility of an exploration step on the next trial, and we found that exploration efficiency was only slightly improved while computation time increased significantly. In the work reported here, we used a greedy approximation, in which we assume that after the current repetition the agent just uses the best known path for future repetitions. This leads to the following formula for the cost C_e of an exploratory path including unknown edge e:

$$C_e = pL_nR + (1-p)(L_f + (R-1)L_o)$$

Note that this formula accounts for the possibility of wasting extra time by exploring, since L_f includes the cost of recovering from a failed exploratory action.

The exploration strategy as described above is *greedy*, in that only one unknown edge is examined on each trip. We improve this greedy method by applying it recursively. After the agent attempts to traverse an unknown edge and adds its new knowledge to the known graph, LWP is applied again to find the best exploratory path between the new current node and B[4].

The full LWP algorithm is given in Figure 2. The agent evaluates the utility of all outstanding exploratory paths (Step 4), and chooses the best such 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. At that point, the agent uses the best known path to get to its current target (Step 7a).

5.2 LBP: Learning 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 LWP algorithm with a *learning before performing* (LBP) algorithm, where the agent attempts to study *all* the edges in the environment, and only then moves so as to carry out its tasks. The exploration is performed using a heuristic backtracking traversal of the graph, starting with the spanning tree given to the agent at the outset. The heuristic used prefers attempting to directly move to a target node or to a node known to be adjacent to a target node. The agent first explores any unknown edges incident on its current

[4] We also experimented with the greedy version of the algorithm, which explores only one unknown edge per task repetition. While the greedy version was more costly, it still dominated other algorithms in most cases. This indicates that the utility-based focus is the main factor in LWP's efficiency.

Algorithm 1 *LWP*(A, B, R)

1. *If* $R = 0$*, terminate;*
2. *Let* v_0 *be the current node of the agent, and* C_{known} *be the cost of completing the remaining* R *tasks using only paths in the known environment;*
3. *Enumerate the set of unknown edges* $\{e^i = (v_1^i, v_2^i)\}$*;*
4. *Evaluate the expected cost* C_e *of each unknown edge* e*;*
5. *Let* $e^* = (v_1^*, v_2^*)$ *be the unknown edge with lowest expected cost* C_{e^*}*;*
6. *If* $C_{e^*} \leq C_{\text{known}}$*, then:*
   (a) *Follow the best known path to get to* v_1^**;*
   (b) *Attempt to move to* v_2^**;*
   (c) *If the traversal succeeded, add* e^* *to the known graph;*
   (d) *Otherwise, note that* e^* *doesn't exist;*
   (e) *If the current node is* 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.*

Figure 2: The LWP interleaved exploration algorithm.

location (in heuristic order), and backtracks when no such unexplored edges exist. We did not use a potentially more efficient search strategy such as Real-Time A* [23], 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 LBP is allowed to explore until the entire environment is known, LWP 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.

Efficiency of LBP

The worst case performance of this algorithm can be roughly bounded by noting that no existing edge is ever traversed more than twice during exploration, once when it is first traversed, and then possibly once during backtracking. However, no non-existent edge is ever attempted more than once. Hence, for random graphs with edge probability p, the average worst-case performance of LBP is

$$W_{LBP} = (1+p)\frac{n(n-1)}{2}$$

We can now bound the average worst-case performance of LBP per task, given the number of task repetitions R, assuming that a path of length α exists. Note that in the worst case, just one task will be accomplished during exploration. Hence, the per task cost for R repetitions is bounded by:

$$T_{LBP} = \frac{W_{LBP} + (R-1)\alpha}{R}$$

For parameters $\alpha = 3$, $n = 100$, $R = 100$, and $p = 0.1$, we have that $T_{LBP} = 57.42$, while the actual per task cost for this scenario incurred by LBP in our experiments (described in Section 6 below) was 4.32. Thus in practice, the algorithm performs far better than these worst case bounds, and hence is a worthy opponent for our LWP algorithm.

5.3 RLWP: Randomized Learning While Performing tasks

We also compared our original algorithm with a simpler algorithm that explores randomly while performing its tasks. This *randomized learning while performing* (RLWP) algorithm is modeled on probabilistic exploration methods used in reinforcement learning (see, eg, [30]). The algorithm generally follows the shortest known path towards its current goal, but if the current node has any unknown neighboring edges, with *exploration probability* p_e RLWP attempts to traverse the best such unknown edge according to the heuristic function used by LBP (see above). 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 RLWP stops exploring when the best known path between A and B is shorter than α.

6 Experiments

6.1 The simulation

In order to compare the above different algorithms we performed simulations on randomly generated graph navigation tasks. The basic parameters for the simulations were the number of nodes n in the graphs to be considered, the edge probability p, the number of task repetitions R, and the stopping criterion α (for LBP and RLWP). For each experimental trial, we report the average results over an ensemble of 100 randomly generated graphs. All of the algorithms were tested on the same ensemble of graphs for each trial.

We considered two types of randomly generated graphs. The first type we call *random graphs*, where edges are generated between pairs of nodes according to the probability p. These graphs are typically very non-planar, and a small amount of exploration can help a lot, if a good shortcut is found. In fact, for sufficiently large random graphs, the longest minimal path in the graph between any two nodes (its *diameter*) is almost always of length 2 (see [10] and Appendix A). We found empirically that for graphs with 100 nodes, the average diameter depended on the edge probability p:

- $p = 0.01$ Average diameter = 4.4
- $p = 0.05$ Average diameter = 2.5
- $p = 0.10$ Average diameter = 2.1

The other type of graph we considered are *triangle graphs*, in which edges are randomly generated based on the Delaunay triangulation of a random point-set in the plane[5]. In these graphs the potential benefit of exploration may be somewhat less, since the likelihood of finding a good shortcut is lower. Indeed, for triangle graphs, the average graph diameter for graphs of 100 nodes was 4.2 for $p \geq 0.05$.

6.2 Random graph results

In these experiments, we compared the algorithms' performance on random graphs (with initially-known spanning trees), which were generated as follows, given graph 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.

[5] We experimented on triangle graphs as well as random graphs since triangle graphs give a model similar to that found in real-world navigation tasks.

R	p =0.01	p =0.05	p =0.1
10	0.94	0.52	0.42
50	0.71	0.38	0.27
100	0.62	0.35	0.26
200	0.56	0.32	0.25

Table 1: Ratio of task performance time of learning while performing (LWP) over the default path cost, in random graphs with $n = 100$ nodes. We compare performance for different edge probabilities p and different numbers of task repetition R.

	$\alpha = 3$			$\alpha = 4$		
R	p =0.01	p =0.05	p =0.1	p =0.01	p =0.05	p =0.1
10	0.03	0.04	0.19	0.03	0.15	0.48
50	0.09	0.13	0.44	0.11	0.42	0.90
100	0.15	0.22	0.61	0.18	0.59	0.78
200	0.24	0.35	0.75	0.29	0.72	0.82

Table 2: Ratio of overall task performance cost for LWP over LBP, for random graphs of size $n = 100$, for different edge probabilities p, number of task repetitions R, and LBP stopping criterion α.

2. Find a minimal spanning tree $T = (V, E_T)$ for this weighted graph.

3. Let $E' = E_T$.

4. 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.

5. Choose, with probability $\frac{1}{n}$ and $\frac{1}{n-1}$, two nodes in the graph as goals, A and B.

6. Output the graph $G = (V, E')$ with initial spanning tree E_T, and goals A and B.

In Table 1, we compare the overall efficiency of LWP for repeated tasks versus the default of using the path given in the initial spanning tree. We see that in all cases, efficiency is improved by using interleaved 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 also increases as we consider 'denser' worlds, with a higher edge probability, since (i) the likelihood of finding a shortcut is higher, and (ii) action failure is less likely, lowering the expected cost of exploring.

We further evaluated the differences between the average (amortized) cost of each task repetition for the three exploration algorithms; detailed results are given below. The differences between the algorithms were tested using χ^2, two-sample t-test, and analysis of paired data. The differences between LWP and the other two algorithms tested significant to a 0.05 confidence level, showing LWP to be more effective than the other two algorithms.

We first compare the results of utility-based interleaved exploration (LWP) versus a priori exploration (LBP), summarized in Tables 2 and 3. In all cases, LWP achieves lower total cost than LBP (ratio < 1).

	$\alpha = 3$			$\alpha = 4$		
n	p =0.01	p =0.05	p =0.1	p =0.01	p =0.05	p =0.1
10	0.98	0.96	0.95	0.98	0.94	0.91
20	0.83	0.80	0.85	0.89	0.88	0.82
30	0.67	0.71	0.82	0.75	0.84	0.85
40	0.55	0.58	0.72	0.61	0.75	0.79
50	0.39	0.46	0.72	0.49	0.72	0.82
60	0.30	0.39	0.63	0.38	0.67	0.79
70	0.25	0.32	0.58	0.29	0.72	0.86
80	0.21	0.28	0.59	0.27	0.61	0.87
90	0.16	0.26	0.50	0.21	0.70	0.86
100	0.14	0.22	0.60	0.18	0.59	0.78

Table 3: Ratio of overall task performance cost for LWP over LBP, for random graphs with number of repeats $R = 100$, for different edge probabilities p, graph sizes n, and LBP stopping criterion α.

	$\alpha = 3$			$\alpha = 4$		
R	p =0.01	p =0.05	p =0.1	p =0.01	p =0.05	p =0.1
10	0.45	0.52	0.72	0.47	0.59	0.82
50	0.47	0.62	0.85	0.52	0.79	0.86
100	0.59	0.75	0.92	0.64	0.87	0.87
200	0.75	0.90	0.97	0.79	0.93	0.88

Table 4: Ratio of overall task performance cost for LWP over RLWP, for random graphs of size $n = 100$, for different edge probabilities p, number of task repetitions R, and RLWP stopping criterion α.

Overall, better results are achieved for LBP with a more conservative stopping criterion ($\alpha = 4$ rather than 3). We also note that a priori exploration is relatively less effective in sparser environments (lower p), since its search strategy will fail attempted traversals many times during exploration, whereas the utility-based interleaved algorithm stops exploring when the chance of finding a useful edge is too low.

Table 2 shows the dependence of the algorithms' performance on the number of task repetitions. As we expect, the advantage of interleaved exploration over a priori exploration is reduced as the number of repetitions grows, since the cost of learning before doing is amortized over more repetitions.

In Table 3 we see the dependence of the algorithms' performance on the size of the environment. Here we see a clear difference between sparse and dense environments. In sparse environments ($p = 0.01$), interleaved exploration becomes more effective relative to a priori exploration as the environment grows larger. In dense environments ($p = 0.1$), however, a priori exploration ($\alpha = 4$) remains relatively effective as the environment grows larger, since it can find a good path between the start and the goal relatively quickly.

In order to understand better the contribution of the utility-based focus of LWP, we also compared its performance against that of RLWP, an interleaved algorithm that explores randomly. We

n	$\alpha = 3$			$\alpha = 4$		
	p =0.01	p =0.05	p =0.1	p =0.01	p =0.05	p =0.1
10	0.99	0.98	0.95	0.98	0.95	0.90
20	0.93	0.91	0.90	0.95	0.92	0.81
30	0.89	0.91	0.94	0.92	0.93	0.87
40	0.83	0.85	0.90	0.86	0.88	0.86
50	0.78	0.83	0.93	0.82	0.89	0.88
60	0.68	0.79	0.92	0.72	0.86	0.83
70	0.67	0.78	0.93	0.71	0.87	0.87
80	0.66	0.78	0.96	0.71	0.87	0.85
90	0.59	0.78	0.92	0.66	0.90	0.89
100	0.58	0.75	0.92	0.63	0.87	0.87

Table 5: Ratio of overall task performance cost for LWP over RLWP, for random graphs with number of repeats $R = 100$, for different edge probabilities p, graph sizes n, and RLWP stopping criterion α.

m	p =0.01	p =0.05	p =0.1
0.5	1.002	0.965	0.994
1.0	**1.000**	**1.000**	**1.000**
1.5	1.016	1.011	1.000
2.0	1.030	1.014	1.000
2.5	1.043	1.016	1.000
3.0	1.053	1.018	1.000

Table 6: Robustness of utility-based interleaved exploration to estimation of p. The table gives the average cost of task completion for greedy LWP using mistaken estimates of p, in random graphs of $n = 100$ nodes with $R = 100$ task repetitions, where the scale is set so the cost using an accurate estimate is 1.0. The estimate of p used by the algorithm was $m \times p$ in each case.

chose an exploration probability for RLWP of 0.3, which overall worked best in our experiments. The results of the comparison of LWP with RLWP are summarized in Tables 4 and 5. Here we see a definite advantage of the utility-based approach over the randomized approach in virtually all cases. As the number of task repetitions increases, however, the advantage of LWP decreases. With more repetitions over which to amortize the cost of exploration, the focus provided by utility evaluation is less crucial. On the other hand, as the size of the environment increases, this focus becomes more and more important, and so LWP's advantage over RLWP becomes more and more pronounced. In dense environments ($p = 0.1$), RLWP performs considerably better, although LWP still dominates in the cases we considered.

6.2.1 Robustness of LWP

Table 6 shows results of experiments we performed to test our method's sensitivity to errors in its estimate of the edge probability p. We generated 100 random graphs of 100 nodes each, for each of the true edge probabilities $p = 0.01, 0.05$, and 0.1. We then ran the simpler, greedy version of LWP (checking just one exploratory path per repetition) on each of the graphs, where the algorithm's estimate of p was $m \cdot p$, for values of m between 0.5 and 3. As the table shows, the difference in execution time for even a 200% error in p is no more than 5%. The algorithm thus appears to be robust to error in estimating p.

6.3 Triangle graph results

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

Triangle graphs 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 [3]).
3. Assign each edge a random weight.
4. Find a minimal spanning tree $T = (V, E_T)$ for this weighted graph.
5. Let $E' = E_T$.
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 $\frac{1}{n-1}$, two nodes in the graph as goals, A and B.
8. Output the graph $G = (V, E')$ with initial spanning tree E_T, 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.

Data comparing the various exploration algorithms for triangle graphs is presented in Tables 7 and 8. The first important result is that exploration (using any method) does not improve much over default performance (using the initial spanning tree). Finding improved paths in triangle

n	LWP/LBP p =0.01	p =0.05	p =0.1	LWP/RLWP p =0.01	p =0.05	p =0.1	LWP/Default p =0.01	p =0.05	p =0.1
10	1.02	0.94	0.91	1.01	0.94	0.90	0.99	0.85	0.79
20	0.88	0.91	0.88	0.97	0.94	0.89	0.94	0.73	0.61
40	0.65	0.65	0.71	0.88	0.86	0.86	0.97	0.62	0.52
50	0.49	0.54	0.63	0.75	0.81	0.84	0.84	0.52	0.47
60	0.44	0.46	0.48	0.75	0.75	0.76	0.89	0.54	0.51
70	0.34	0.39	0.39	0.71	0.74	0.75	0.86	0.51	0.50
80	0.30	0.34	0.34	0.68	0.78	0.77	0.85	0.59	0.58
90	0.28	0.27	0.28	0.70	0.73	0.74	0.86	0.59	0.60
100	0.24	0.22	0.22	0.70	0.67	0.67	0.88	0.63	0.63

Table 7: Efficiency ratios for LWP over LBP, RLWP, and the default path, for triangle graphs with different numbers of nodes ($R = 100$, $\alpha = 4$).

R	LWP/LBP p =0.01	p =0.05	p =0.1	LWP/RLWP p =0.01	p =0.05	p =0.1	LWP/Default p =0.01	p =0.05	p =0.1
10	0.03	0.04	0.05	0.46	0.83	0.93	1.19	1.65	1.85
50	0.13	0.12	0.13	0.59	0.60	0.62	1.09	0.80	0.84
100	0.21	0.18	0.19	0.70	0.67	0.67	0.88	0.63	0.63
200	0.31	0.27	0.27	0.80	0.78	0.78	0.72	0.51	0.51

Table 8: Efficiency ratios for LWP over LBP, RLWP, and the default path, for triangle graphs with different numbers of task repetitions ($n = 100$, $\alpha = 4$).

graphs is more difficult, because true shortcuts (ones that bypass large portions of the graph) do not exist. This means that the interleaved algorithms' essentially local nature may preclude them from finding some good paths, while LBP must search much longer to find a short path in its initial exploration phase.

Since $\alpha = 4$ gave the best performance for LBP and RLWP, we present results for that value, and thus conservatively estimate the relative usefulness of LWP. It appears that LWP is never much worse than LBP and RLWP for triangle graphs, and is sometimes much better. The difference is most pronounced in comparison with LBP, where we find a considerable gain in efficiency in using LWP over LBP as the size of the environment increases (see Table 7). We also saw such a trend for random graphs, though the effect was much less. The difference between the two interleaved algorithms, LWP and RLWP, is less pronounced, however; using a utility-based formulation instead of random local exploration may not help much in some planar environments.

7 Multiple Repeated Tasks

In the previous sections, we described and evaluated algorithms for combining exploration and task achievement in a partially-known 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. If we model the environment as a graph

	LWP/LBP				LWP/RLWP				LWP/Default			
	R				R				R			
# goals	100	500	1000	2000	100	500	1000	2000	100	500	1000	2000
2	0.62	0.89	0.95	0.98	0.88	0.97	1.01	1.01	0.34	0.31	0.30	0.30
4	0.32	0.56	0.64	0.71	0.71	0.80	0.80	0.82	0.53	0.36	0.34	0.33
6	0.37	0.58	0.65	0.69	0.79	0.82	0.81	0.78	0.69	0.41	0.37	0.34
8	0.40	0.58	0.66	0.69	0.82	0.83	0.81	0.79	0.84	0.44	0.38	0.34
10	0.43	0.61	0.68	0.71	0.79	0.83	0.82	0.80	0.93	0.47	0.40	0.36
20	0.47	0.74	0.78	0.79	0.71	0.94	0.91	0.86	1.15	0.61	0.49	0.41
30	0.49	0.89	0.88	0.86	0.70	1.07	0.99	0.92	1.17	0.73	0.55	0.45
50	0.50	0.96	1.02	0.96	0.67	1.12	1.12	1.02	1.24	0.81	0.64	0.51

Table 9: Results for multiple repeated goals in random graphs. Number of nodes $n = 100$, and edge probability $p = 0.05$.

$G = (V, E)$, instead of the agent repeatedly moving back and forth between two designated nodes (as above), each time the agent reaches its current goal node, its next goal node is selected randomly according to some probability distribution over V. We assume this 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 assumed a uniform probability distribution over a subset S of V.

We generalized the exploration algorithms described above to the case of multiple goals in a straightforward manner. For the learning before performing (LBP) and the randomized learning while performing (RLWP) algorithms, the only change necessary is in the termination criterion, where exploration is terminated if the *average* shortest path length between pairs of possible goals is less than the threshold α. The utility-based learning while performing (LWP) 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 known path length over all pairs of possible goals, for each unknown edge.

In general, we expect the efficiency of interleaved exploration versus a priori exploration to decrease as the number of different goals increases, since the focusing that interleaved exploration provides will be less useful. Our results are presented in Tables 10 and 9.

Surprisingly, we find that, at least in dense environments ($p = 0.1$, Table 10), LWP performs considerably better than LBP, even for large numbers of goals. The difference decreases somewhat as the number of tasks increases, but remains significant. In sparser environments ($p = 0.05$, Table 9), however, this improvement is reduced, and nearly disappears for large numbers of repetitions. We conjecture that in dense environments, there is a great deal of overlap between good paths for different goals, and so the focus of interleaved exploration can still be used to advantage. The utility-based focus, however, seems less critical, at least over the long run, since the difference between LWP and RLWP is quite small for larger numbers of repetitions. Interestingly, the ratio of LWP/RLWP seems to increase with the number of repetitions up to a point (between 500 and 1000 iterations) and then begins to decrease. It is not yet clear to us why this should occur.

Finally, we note that in all cases, LWP performs better than the default of using an initial

	LWP/LBP				LWP/RLWP				LWP/Default			
	R				R				R			
# goals	100	500	1000	2000	100	500	1000	2000	100	500	1000	2000
2	0.94	0.91	0.90	0.90	0.96	0.92	0.89	0.91	0.28	0.26	0.25	0.25
4	0.25	0.45	0.53	0.58	0.82	0.75	0.73	0.72	0.40	0.29	0.28	0.27
6	0.29	0.47	0.53	0.57	0.86	0.74	0.69	0.69	0.50	0.32	0.29	0.27
8	0.30	0.48	0.53	0.56	0.91	0.77	0.71	0.67	0.57	0.34	0.30	0.27
10	0.34	0.51	0.56	0.58	0.90	0.78	0.71	0.68	0.65	0.36	0.31	0.28
20	0.43	0.61	0.65	0.65	0.97	0.89	0.81	0.73	0.92	0.47	0.39	0.33
30	0.45	0.69	0.73	0.70	0.93	0.99	0.90	0.79	0.97	0.54	0.43	0.36
50	0.46	0.82	0.84	0.81	0.90	1.12	1.02	0.90	1.04	0.64	0.51	0.42

Table 10: Results for multiple repeated goals in random graphs. Number of nodes $n = 100$, and edge probability $p = 0.1$.

spanning tree. As expected, the improvement increases with the number of repetitions.

8 Future Work

The work described in this paper makes several strong modeling assumptions. One is that the agent begins knowing a spanning tree of the graph to be explored. While this assumption is not unreasonable, it might be relaxed either by allowing the agent to ask someone how to proceed (for some known cost), or by having the agent know nothing and proceed randomly until a path is found. Both of these can be addressed in our utility-based formulation—asking for help becomes another action whose expected utility can be estimated, and the expected cost of random exploration can be estimated from experience. Similarly, if the costs of different edges may vary, a number of different modeling options are available. If the geometric distance between two nodes is known, this information can be used to estimate the actual path distance between them in the graph. More generally, if some probability distribution can be established for the path distance between any pair of nodes, the agent can use the expected path distance in calculating the expected utility of an exploratory action. Such distributions may be learned through experience, and in practice may depend on features of the nodes considered. For example, an agent on the Internet would learn to estimate the 'distance' between two `.edu` sites as lower than that between a `.edu` site and a `.il` site. The final assumption used by all the algorithms described in this paper is that the agent knows all the nodes in the environment at the start. This assumption is reasonable for some environments (such as the Internet), but less reasonable for others (such as a mobile robot). Relaxing this assumption involves changing the problem model to one in which the agent is only aware of its current node and its outgoing arcs. We are currently applying our utility-based exploration algorithm to models of this type.

9 Discussion

The LWP algorithm presented above uses a greedy estimate of exploration utility in order to explore a partially-known graph during performance of repeated tasks. Our results for a several different types of graphs show interleaved exploration to be generally superior to a good a priori exploration algorithm (LBP). We further have shown that LWP usually gives better performance than a randomized interleaved exploration algorithm (RLWP), demonstrating the importance of explicitly considering the expected utility of exploration. Utility-based interleaved 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 LWP can give significant efficiency improvements also for the case of 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 a simple utility-based interleaved exploration strategy can sometimes achieve a significant improvement in agent performance over other exploration strategies.

References

[1] S. Albers and M. R. Henzinger. Exploring unknown environments. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 416–425, 1997.

[2] S. Argamon-Engelson, S. Kraus, and S. Sina. Utility-based on-line exploration for repeated navigation in an embedded graph. *Artificial Intelligence*, 102(1–2):267–284, 1998.

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

[4] 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.

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

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

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

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

[9] 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.

[10] B. Bollobas. *Random Graphs.* Academic Press, London, 1985.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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.

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

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

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

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] R. Vaknin. Robotic mapping during task execution. Master's thesis, Bar-Ilan University, 1999. In Hebrew.

A Occurrence of short paths in random graphs

In this appendix we investigate the probability of short paths between target nodes A and B occurring in random graphs. Random graphs are parameterized by number of nodes n and edge probability p. (For simplicity, we ignore here the existence of a spanning tree.) We derive the probability $Q(n,p;\leq k)$ that at least one path of length less than or equal to k (for $k = 3, 4$) between nodes A and B occurs in a random graph with n nodes and edge probability p as follows.

First, consider the probability of a path of length 1 occurring between A and B. Either the edge AB exists or it doesn't, thus:

$$Q(n,p;=1) = p$$

Now, consider the probability of a path of length less than or equal to 2. If a path of length 1 occurs, then such a path exists. If not, we consider all $n-2$ possible paths of length 2 from A to B. The probability of each such path occurring is p^2, since each of its edges occurs independently with probability p. Hence:

$$\begin{aligned} Q(n,p;\leq 2) &= Q(n,p;=1) + [1 - Q(n,p;=1)]Q(n,p;=2|\neq 1) \\ &= p + (1-p)[1-(1-p^2)^{n-2}] \end{aligned}$$

where $Q(n,p;=2|\neq 1)$ denotes the probability (in random graphs of n nodes with edge probability p) of a path of length 2 occurring, given that no paths of length 1 occur. We generalize this notation below in the obvious way.

It is clear from $Q(n,p;\leq 2)$ that the probability of a path of length ≤ 2 goes to 1 as n goes to infinity (also see [10]).

Next, consider the probability of a path of length less than or equal to 3. As above, we formulate:

$$\begin{aligned} Q(n,p;\leq 3) &= Q(n,p;=1) \\ &+ [1-Q(n,p;=1)][Q(n,p;=2|\neq 1) + [1-Q(n,p;=2|\neq 1)]Q(n,p;=3|\neq 1 \wedge \neq 2)] \end{aligned}$$

The difficulty here is determining the probability of a path of length three occurring given that no paths of length 1 or 2 occur, $Q(n,p;=3|\neq 1\wedge \neq 2)$. The conditioning event destroys the independence of the edge probabilities for edges incident on A or B. Consider a potential such edge AC. The probability of this edge, given that no path of length 2 from A to B exists, is:

$$\begin{aligned} P(AC) &= \tfrac{P(AC,\neg CB)}{1-P(AC,CB)} \\ &= \tfrac{p(1-p)}{1-p^2} \\ &= \tfrac{p}{1+p} \end{aligned}$$

By symmetry, the edges incident on A and B have equal probability, $q = P(AC) = P(AD) = P(CB) = P(DB) = \frac{p}{1+p}$. Considering a potential path $ACDB$ of length 3, the probability of the middle edge CD is independent of the occurrence of paths of length 1 or 2, so we have that the probability of such a path is q^2p. Hence, quantifying over the $(n-2)(n-3)$ potential paths:

$$Q(n,p;=3|\neq 1\wedge \neq 2) = 1-(1-q^2p)^{(n-2)(n-3)}$$

and thus

$$\begin{aligned} Q(n,p;\leq 3) &= Q(n,p;=1)+ \\ &\quad [1-Q(n,p;=1)][Q(n,p;=2|\neq 1)+[1-Q(n,p;=2|\neq 1)](1-(1-q^2p)^{(n-2)(n-3)})] \\ &= p+(1-p)[(1-(1-p^2)^{n-2})+((1-p^2)^{n-2})(1-(1-q^2p)^{(n-2)(n-3)})] \end{aligned}$$

For paths of lengths less than or equal to 4, we again proceed similarly, with:

$$\begin{aligned} Q(n,p;\leq 4) &= Q(n,p;=1)+ \\ &\quad (1-Q(n,p;=1))[\ Q(n,p;=2|\neq 1)+ \\ &\qquad (1-Q(n,p;=2|\neq 1))\,[\ Q(n,p;=3|\neq 1\wedge \neq 2)+ \\ &\qquad\quad (\,(1-Q(n,p;=3|\neq 1\wedge \neq 2))\times \\ &\qquad\quad Q(n,p;=4|\neq 1\wedge \neq 2\wedge \neq 3)\,)]] \end{aligned}$$

In order to calculate $Q(n,p;=4|\neq 1\wedge \neq 2\wedge \neq 3)$, we must, as above, calculate the probability q_1 of an edge from A or B to another node C, given that no paths ≤ 3 exist between A and B, as well as the probability q_2 of an edge between two nodes C and D other than A and B, under the same condition. We do so as follows:

$$\begin{aligned} q_1 &= Pr(AC|\not\leq 3) \\ &= \tfrac{Pr(AC\wedge\not\leq 3)}{Pr(\not\leq 3)} \\ &= \tfrac{Pr(AC\wedge\not\leq 3)}{1-Pr(\leq 3)} \\ &= \tfrac{Pr(AC\wedge\not\leq 3)}{1-Q(n,p;\leq 3)} \end{aligned}$$

Now, let $r = Q(n-1,p;\not\leq 3)$ be the probability of no paths of length less than 3 between A and B in $G\backslash C$. These are independent of the occurrence of AC. Further, consider all $n-3$ nodes

$D_i \notin \{A, C, B\}$—we require that no path of length 2 from C to B via any D_i exist. Hence:

$$\begin{aligned} q_1 &= \frac{Pr(AC \wedge \neg CB \wedge \bigwedge_i \neg(CD_i \wedge D_iB))r}{1-Q(n,p;\leq 3)} \\ &= \frac{Pr(AC)Pr(\neg CB)Pr(\bigwedge_i \neg(CD_i \wedge D_iB))rr_1}{1-Q(n,p;\leq 3)} \\ &= \frac{p(1-p)(1-p^2)^{n-3}(1-Q(n-1,p;\leq 3))}{1-Q(n,p;\leq 3)} \end{aligned}$$

Similarly:

$$\begin{aligned} q_2 &= Pr(CD|\not\leq 3) \\ &= \frac{Pr(CD \wedge \not\leq 3)}{Pr(\not\leq 3)} \\ &= \frac{Pr(CD \wedge \not\leq 3)}{1-Pr(\leq 3)} \\ &= \frac{Pr(CD \wedge \not\leq 3)}{1-Q(n,p;\leq 3)} \end{aligned}$$

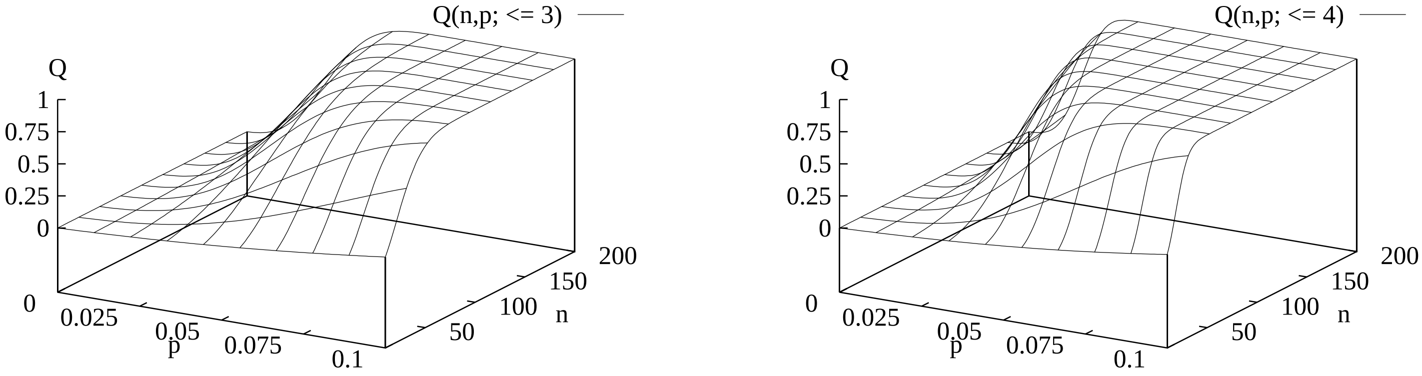
Now, let $s = Q(n-2, p; \not\leq 3) = 1 - Q(n-2, p; \leq 3)$ be the probability of no paths of length 3 between A and B in $G \backslash \{C, D\}$. Then:

$$\begin{aligned} q_2 &= \frac{Pr(CD \wedge \neg(AC \wedge DB) \wedge \neg(BC \wedge DA))s}{1-Q(n,p;\leq 3)} \\ &= \frac{Pr(CD)Pr(\neg(AC \wedge DB)Pr(\neg(BC \wedge DA))s}{1-Q(n,p;\leq 3)} \\ &= \frac{p(1-p^2)^2(1-Q(n-2,p;\leq 3))}{1-Q(n,p;\leq 3)} \end{aligned}$$

Hence, the probability of a particular path of length 4, given no paths of lengths ≤ 3, is $q_1^2 q_2^2$, thus, since there are $(n-2)(n-3)(n-4)$ possible paths of length 4:

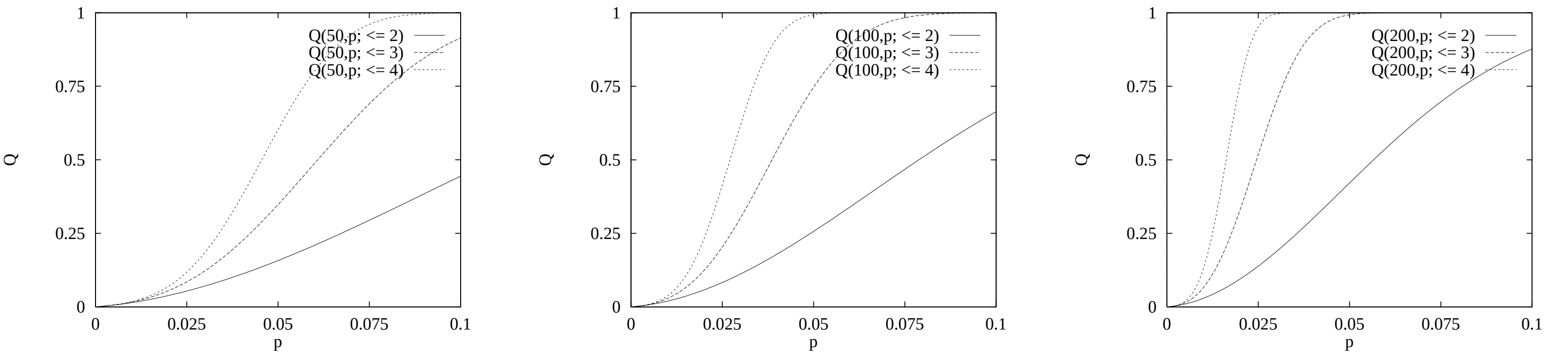
$$\begin{aligned} Q(n,p;\leq 4) \;=\; & Q(n,p;=1)+ \\ & (1-Q(n,p;=1))\times \\ & [Q(n,p;=2|\neq 1) + (1-Q(n,p;=2|\neq 1))[\; Q(n,p;=3|\neq 1 \wedge \neq 2)+ \\ & \qquad (1-Q(n,p;=3|\neq 1 \wedge \neq 2))\times \\ & \qquad (1-(1-q_1^2 q_2^2)^{(n-2)(n-3)(n-4)})]] \end{aligned}$$

In Figures 3 and 4, we plot the functions $Q(n, p; \leq k)$, for $k = 2, 3, 4$. It is clear that the number of nodes n has a great effect. For $n = 200$, the probability of a path length ≤ 4 is nearly 1, even for $p = 0.025$. In particular, note that $Q(n, p; \leq 4)$ is often much larger than $Q(n, p; \leq 3)$, i.e, there are many graphs with paths of length 4, but not of length 3. This explains why a stopping criterion of $\alpha = 4$ for LBP often performs better than $\alpha = 3$, since the lower α criterion will often fruitlessly explore the entire graph, even after finding a path of length four.



(a) (b)

Figure 3: The functions (a) $Q(n,p;\leq 3)$ and (b) $Q(n,p;\leq 4)$, plotted as functions of p and n.



(a) (b) (c)

Figure 4: The functions $Q(n,p;\leq 2)$, $Q(n,p;\leq 3)$, $Q(n,p;\leq 4)$, plotted as functions of p, for (a) $n=50$, (b) $n=100$, (c) $n=200$.