PRESENTs: Probabilistic REsource-SEarch NeTworks*

Qing Guo
Department of Computer Science
University of Illinois at Chicago
qguo@cs.uic.edu

Ouri Wolfson
Department of Computer Science
University of Illinois at Chicago
wolfson@cs.uic.edu

ABSTRACT

We consider a model in which there are spatially located static resources on a road network, and a mobile agent. The agent looks to obtain one of the resources, and has no knowledge of exact availability of the resources; only probabilistic information is available. We develop a novel and efficient algorithm that guides the agent through a road network in order to find the desired resource at minimal cost.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

Keywords

Spatio-temporal databases, probabilistic data, routing

1. INTRODUCTION

In recent work, with the help of smartphones, a taxi driver is able to locate passengers [2]; and with the help of wireless embedded sensors, a vehicle is able to locate available on-street parking spaces (e.g. http://sfpark.org/). In these examples, the taxi driver and the vehicle are the agents, and the passengers and parking spaces are resources that the agents attempt to find. Other examples of resources include Electric-Vehicles charging stations, rental bikes, share-bike parking slots, and packages for pick-up and delivery.

The exact number of available resources per location is usually unknown to the agent. Oftentimes, only uncertain data is available. This situation is common for data collection using crowdsourcing methods, where only some of the agents are used for crowdsourcing (see e.g. Xu et al. [4]).

In this work we assume that there is a graph representing a road network, and that the resources are located on the edges of the graph. Each edge is associated with a cost of traversing it (e.g. the travel time), and a probability of having at least one available resource. We address the problem of efficiently searching for a resource in the road network. Specifically, *the problem is to find an optimal search-path,*[1] *and to do so by an efficient algorithm.*

What is the meaning of an optimal search-path for a resource? An approach introduced previously to answer this question is the Probability Maximization (PM) algorithm [5]. This algorithm computes the total probability of finding a resource for any given path of length (cost) M or less from a starting location. Then a path with the maximum probability is picked as the optimal path.

This approach has several drawbacks. First, every infinite path has probability 1 of finding a resource.[2] Thus, all unbounded search-paths have identical probabilities. Therefore, the PM approach does not work when the agent is willing to search as long as it takes in order to find a resource (e.g. a taxi cab searching for a customer at 9 am). Second, consider the road network example in Figure 1. It consists of a network of two parallel edges, each with probability 0.5. One edge takes 15 minutes to traverse, and the other takes 5 minutes. If the bound on the search is 15 minutes, then PM is indifferent between the two paths, and may choose either, although the shorter path is clearly superior.

Thus, in this paper we introduce an alternative method, the Minimum Expected Cost (MEC), to efficiently search for a resource.[3] It produces a path of minimum expected cost. Consider again Figure 1. Observe that the shorter path has a lower expected cost, thus it will be preferred by MEC over the other. MEC produces such a path efficiently, for bounded and unbounded (i.e. infinite) paths. And experimental results with parking and taxi cab data indicate that MEC is superior to PM in terms of resource search-time.

2. ASSUMPTIONS AND NOTATION

A *road network* is a directed graph; the vertices are the intersections of the roads, and the edges are the road segments connecting the intersections. Assume that there are

*This work was supported in part by NSF grants DGE-0549489, IIS-1213013 and CCF-1216096, and by USDOT NURAIL Center.

SIGSPATIAL'15, November 03-06, 2015, Bellevue, WA, USA
 ISBN 978-1-4503-3967-4/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2820783.2820866.

[1]Observe that in a search an edge can be visited multiple times, because an unavailable edge (i.e. an edge that does not have an available resource) may become available later.
[2]Assume that the infinite path is $\{e_0 \rightarrow e_1 \rightarrow \cdots\}$, then the overall probability of getting at least one available resource along it, is $p = 1 - \prod_{i=1}^{\infty}(1 - p_i) = 1$, where p_i is the probability of finding a resource on edge e_i
[3]Our model with unbounded search is analogous to Markov Decision Processes (MDPs) [3] in infinite time horizons with discounted rewards. The difference is that the edge costs in our model are not discounted over time.

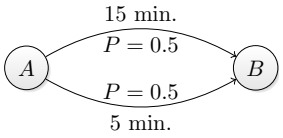

Figure 1: An example of comparing the algorithms.

n vertices in the road network, denoted by v_i, where $i = 1, 2, \ldots, n$, and let edge e_{ij} be the road segment between vertices v_i and v_j. Let the *travel cost* from v_i to v_j, denoted as c_{ij}, be the cost of e_{ij}. The cost may represent *travel time* to traverse the edge, *usage cost* of a resource on this edge, or a combination of both; e.g., in the case of parking, usage cost is the time to walk from the edge to the final destination.

Let p_{ij} denote the *probability* of edge e_{ij} being available ($0 < p_{ij} < 1$), i.e. the probability that at least one resource is available at e_{ij} at arbitrary point in time during some time interval, e.g. 5 pm to 6 pm. Consequently, $1 - p_{ij}$ is the probability that no resource is available on e_{ij}. We assume that p_{ij} is estimated from historical or partial data.

A *search-path*, is simply a path in the road network. It is provided by MEC, PM, or other algorithms to the agent; it may be *bounded* by a constant K representing the maximum number of edges in the search, or it may be *unbounded*. An unbounded search is requested by an agent willing to continuously traverse the network until obtaining a resource.

After receiving the search-path, the agent follows it. While following a search-path, if the usage cost is 0, then the agent will obtain the first resource that it finds. If the usage cost is higher than 0, then when finding a resource r the agent will compare its usage cost with the expected cost of the rest of the path, and only obtain r if the former is lower.

The *cost* of the search depends on whether or not the search is successful. If the search is *successful*, then the cost of the search is the total cost of the edges traversed (i.e. the prefix of the search-path), plus the usage cost of the obtained resource, if any. For the rest of the paper, we assume that the usage cost is 0. If the search is *unsuccessful*, assume that it ends at vertex v_i, the last vertex on the search-path. Then the cost of the search is the total cost of the edges on the path, plus some constant β_i which denotes the additional penalty for not finding any resource after terminating the search at node v_i. For example, in the parking search application, if no parking space is found, then β_i is the cost of traveling from v_i to a private garage and parking there. If the search represents a taxi driver cruising for customers, β_i represents the cost of returning home empty-handed.

3. EXPECTED COST OF A SEARCH

We define the expected cost of a search in a recursive (stepwise) fashion. In this definition, the expected cost of a search is defined by the cost of its first edge and the expected cost of the sub-path without the first edge. This is for the convenience of presenting the Bellman equation to compute a path with the minimum expected cost (see Section 4).

Denote the expected cost of search-path $\{v_0 \to v_1 \to \cdots \to v_K\}$ as $C_{\{v_0 \to v_1 \to \cdots \to v_K\}}$, then we have

$$\begin{aligned} & C_{\{v_0 \to v_1 \to \cdots \to v_K\}} \\ =& p_{01} c_{01} + (1 - p_{01}) \left(c_{01} + C_{\{v_1 \to \cdots \to v_K\}} \right) \\ =& c_{01} + (1 - p_{01}) C_{\{v_1 \to \cdots \to v_K\}}. \end{aligned} \tag{1}$$

The second line of Equation (1) has two terms. The first is the cost of finding the resource on the first edge of the path, and ending the search; the second is the cost of continuing the search on the rest of the path. Similarly, $C_{\{v_1 \to \cdots \to v_K\}}$, $\ldots$, $C_{\{v_{K-1} \to v_K\}}$ can be expanded recursively.

Observe that without the penalty β_K, the minimum expected cost of a path is zero, given by a 0-length path.

4. BOUNDED SEARCH OPTIMIZATION

In this section we show how to find an *optimal path* of length at most K, i.e. a path with an expected cost that is minimum among all paths with length K or smaller.

Overall, the minimum expected cost of all paths with length $k \in \{1, \ldots, K\}$ from node v_i, is computed based on the minimum expected costs of all possible sub-paths with length $k-1$ starting from v_i's possible successor nodes. This computation relies on the recursive definition of the expected cost of a path as in Equation (1).

For a natural number $k \leq K$, let the *k-step look-ahead from node v_i* be the set of all paths starting from v_i with length k or less. Denote by C_i^k the minimum expected cost of a path in the k-step look-ahead from node v_i. Assume that v_j is an immediate successor of v_i. Denote by C_{ij}^k the expected cost of a path in the k-step look-ahead from node v_i, for which: 1) the first two vertices are v_i and v_j, and 2) the expected cost of the remaining sub-path of length $k-1$ or less is the minimum among all paths of length $k-1$ from v_j. Now we indicate how to compute C_i^k and C_{ij}^k recursively:

$$C_i^k = \begin{cases} \min\limits_{1 \leq j \leq n, \text{ s.t. } e_{ij} \text{ exists}} \left\{ C_{ij}^k, \beta_i \right\}, & \text{if } k > 0. \\ \beta_i, & \text{if } k = 0. \end{cases} \tag{2}$$

$$next_i^k = \begin{cases} \arg\min\limits_{1 \leq j \leq n, \text{ s.t. } e_{ij} \text{ exists}} \left\{ C_{ij}^k \right\}, & \text{if } C_i^k \neq \beta_i. \\ -1, & \text{if } C_i^k = \beta_i. \end{cases} \tag{3}$$

$$C_{ij}^k = c_{ij} + (1 - p_{ij}) C_j^{k-1}. \tag{4}$$

We call a pair of v_i and k a *decision point*. The reason is that at each pair (v_i, k), the computation decides which direction to choose next in order to minimize future costs. The above equations are explained in detail as follows:

Equation (2): When the agent is at v_i, and is allowed to search for k more edges, then the agent should choose a next node v_j such that the expected cost of the remaining path is minimum. If the penalty for terminating the search β_i is smaller than the expected cost of the remaining path, or if $k = 0$, i.e. no further search is allowed, then the cost is β_i and the search is concluded.

Equation (3): The index of the next node v_j that minimizes the cost of a path from v_i with k-step look-ahead is recorded as $next_i^k$. If no further search is needed, then let $next_i^k = -1$. We call $next_i^k$ the *optimal action* for the agent at decision point (v_i, k). In other words, when the agent is at node v_i and has k more edges to search, then $next_i^k$ is the index of the next node that the agent should proceed to in order to minimize the expected cost of the remaining path.

Equation (4): The expected cost for v_i, given v_j as the next node, assuming that the agent will follow the sub-path with minimum expected cost from v_j. (See Equation (1)).

Here is how to extract the optimal path from the resulting $next_i^k$ of this computation. Let v_0 be the starting node, and

$l_1 = next_0^K, l_2 = next_{l_1}^{K-1}, \ldots$, until the first k_0 such that $next_{l_{k_0}}^{K-k_0} = -1$. Then the path starting at v_0 that always minimizes future expected costs is $\{v_0 \to v_{l_1} \to \cdots \to v_{l_{k_0}}\}$.

Although Equations (2) and (4) define a recursive relationship, each C_i^k only needs to be computed once. In other words, they can be regarded as the Bellman equation [3] in a dynamic programming method for computing the minimum expected cost of any path with length K or less. Thus,

THEOREM 1. *The expected cost C_i^K computed by Equations (2), (3), and (4) is the minimum among all paths with length K or less, starting at vertex v_i. Furthermore, C_i^K can be computed in polynomial time $O(ndK)$, where d is the maximum degree of a node.*

PROOF SKETCH. This can be easily proven by induction on K. Specifically, the minimum expected cost C_i^k can be computed in polynomial time given $\{C_j^{k-1} : j = 1, \ldots, n\}$. Therefore, one can compute $\{C_i^1 : i = 1, \ldots, n\}$, and then $\{C_i^2 : i = 1, \ldots, n\}$, until $\{C_i^K : i = 1, \ldots, n\}$. □

As a result, the minimum expected cost from any node, with any time-step look-ahead (up to K), is computed.

5. UNBOUNDED SEARCH OPTIMIZATION

In this section we provide an efficient algorithm to compute the minimum expected cost of an unbounded search, and find the path of such a search.

Analysis of the minimum expected cost of an unbounded search is based on the concept of *contraction* [3], defined as follows. Assume that $\{\boldsymbol{U_k} : k = 1, 2, \ldots\}$ is a set of vectors, and $\boldsymbol{B}$ is an operator (e.g. function). Assume that $\boldsymbol{B}$ defines a sequence $\boldsymbol{U_{k+1}} = \boldsymbol{BU_k}$. If there exists $0 < \gamma < 1$, such that $\forall k, k' \in \{0, 1, \ldots\}, ||\boldsymbol{U_{k+1}} - \boldsymbol{U_{k'+1}}|| = ||\boldsymbol{BU_k} - \boldsymbol{BU_{k'}}|| \leq \gamma ||\boldsymbol{U_k} - \boldsymbol{U_{k'}}||$, then $\boldsymbol{B}$ is a *contraction*.[4] An important property of a contraction is that it has a single fixpoint, and repeatedly applying the contraction will lead to that fixpoint in the limit [3]. In the resource search problem, consider the vector $\boldsymbol{C^k} = \{C_1^k, \ldots, C_n^k\}$ of the minimum expected costs as the vector $\boldsymbol{U_k}$, and Equations (2) and (4) as the operator $\boldsymbol{B}$. Then Lemma 1 indicates that $\boldsymbol{B}$ is a contraction, and the resulting Theorem 2 indicates that the minimum expected cost of an unbounded search converges:

LEMMA 1. *In Equations (2) and (4), $\forall k, k' \in \{0, 1, \ldots\}$, $||\boldsymbol{C^{k+1}} - \boldsymbol{C^{k'+1}}|| \leq \gamma ||\boldsymbol{C^k} - \boldsymbol{C^{k'}}||$, where $\gamma = 1 - p_{min}$.*

PROOF SKETCH. Observe that for any two real sequences $\{a_i : i = 1, \ldots, n\}$ and $\{b_i : i = 1, \ldots, n\}$:

$$\left| \min_{1 \leq i \leq n} \{a_i\} - \min_{1 \leq i \leq n} \{b_i\} \right| \leq \max_{1 \leq i \leq n} \{|a_i - b_i|\}.$$

Then, the inequality in the lemma can be proven by applying Equations (2) and (4) and the above inequality. □

THEOREM 2. *The minimum expected costs $\boldsymbol{C^k}$ defined by Equations (2) and (4) converge to a fixpoint when $k \to \infty$.*

Algorithm 1 (MEC) computes the minimum expected costs and optimal paths of unbounded searches. It builds the minimum expected costs bottom-up by Equations (2), (3), and (4) using dynamic programming for increasing k, until the error is bounded, i.e., until k is large enough so that the minimum expected costs C_i^k and C_i^{k-1} are close enough for every node v_i. When implementing the algorithm, there is no need to store the costs and indices for every k in each iteration; only the vectors $\boldsymbol{C^k}$ and $\boldsymbol{C^{k-1}}$ need to be stored.

[4]The distance between vectors is measured by *max norm*. The max norm of $\boldsymbol{V} = \{V_1, \ldots, V_n\}$ is $||\boldsymbol{V}|| = \max_{1 \leq i \leq n} \{|V_i|\}$.

Algorithm 1 Minimum Expected Cost (MEC) Algorithm

Input: Network $\langle E, V \rangle$, $\{c_{ij}\}$, $\{\beta_i\}$, $\{p_{ij}\}$, error bound ϵ
Output: $\{C_i^k\}$, $\{next_i^k\}$

```
for i = 1, 2, ..., n do
    C_i^0 ← β_i
end for
error ← 1 + ε
k ← 1
while error > ε do
    for i = 1, 2, ..., n do
        C_i^k ← ∞
        for all v_j s.t. e_ij exists do
            C_ij^k ← c_ij + (1 − p_ij) C_j^{k−1}
            if C_ij^k < C_i^k then
                C_i^k ← C_ij^k
                next_i^k ← j
            end if
        end for
        if β_i < C_i^k then
            C_i^k ← β_i
            next_i^k ← −1
        end if
    end for
    error ← ||C^k − C^{k−1}||
    k ← k + 1
end while
```

Infinite Path Representation. Observe that MEC computes the minimum expected costs, and the optimal unbounded search, which may be an infinite path. Now we explain how an infinite path is represented by finite notation. The key is that whenever the agent arrives at the same node, it takes the same action, i.e. proceed to the same successor, or terminate the search. Intuitively, whenever the agent arrives at this node, it considers all the unbounded paths, and the set of unbounded paths from a given node is identical, regardless of how many times the agent arrives at it. This means that $next_i^k$ stays the same for $k \to \infty$. Denote that $next_i = \lim_{k \to \infty} next_i^k$. We call the vector $\{next_i : i = 1, \ldots, n\}$ an *optimal policy* in an unbounded search.

Note that, for a finite search with length K or less, the above argument does not hold. That is, for $k, k' \leq K$ ($k \neq k'$), the optimal actions $next_i^k$ and $next_i^{k'}$ may be different.

Complexity of MEC. The complexity of MEC is dependent on the number of iterations in the while loop. Denote this number by N. In turn, N depends on the given local error bound ϵ via a global error bound ϵ_0. The *local error bound* ϵ bounds the error between two iterations of the while loop in MEC; the *global error bound* ϵ_0 measures the distance between the current value of $\boldsymbol{C^k}$ and its tight upper bound (denoted by $\boldsymbol{C} = \lim_{k \to \infty} \boldsymbol{C^k}$).

To obtain the relationship between N and ϵ_0, we first give a bound on the expected cost of *any* unbounded search-path:

LEMMA 2. *The expected cost of unbounded path $\{v_0 \to$*

$v_1 \to \cdots\}$ *is bounded by the following:*

$$C_{\{v_0 \to v_1 \to \cdots\}} \leq \frac{c_{max}}{p_{min}},$$

where $c_{max} = \max_{all\ e_{ij}} \{c_{ij}\}$, *and* $p_{min} = \min_{all\ e_{ij}} \{p_{ij}\}$.

PROOF SKETCH. By expanding the recursion in Eq. (1),

$$C_{\{v_0 \to v_1 \to \cdots\}} = \lim_{K \to \infty} \left[\sum_{k=1}^{K} \left(c_{k-1,k} \prod_{j=1}^{k-1} (1 - p_{j-1,j}) \right) + \beta_K \prod_{j=1}^{K} (1 - p_{j-1,j}) \right] \leq \frac{c_{max}}{p_{min}} \quad (5)$$

The last inequality results from the fact that in the limit the second term becomes 0, and the first term is bounded by a sum (of a geometric series) that tends to c_{max}/p_{min}. □

Note that, the limit expression of Equation (5) is a (standard) non-recursive definition of the expected cost of a path.

The following lemma specifies the relationship between N and ϵ_0, and gives an idea of how fast the convergence is.

LEMMA 3. *Let* N *denote the number of iterations in the while loop of MEC. For a given global error bound* ϵ_0, *if* $N \geq \left(\log \frac{\beta_{max} + c_{max}/p_{min}}{\epsilon_0}\right) / \left(\log \frac{1}{1-p_{min}}\right)$, *then* $||\boldsymbol{C^N} - \boldsymbol{C}|| \leq \epsilon_0$.

PROOF SKETCH. By Lemma 2, the maximum initial error is $||\boldsymbol{C^0} - \boldsymbol{C}|| \leq ||\boldsymbol{C^0}|| + ||\boldsymbol{C}|| \leq \beta_{max} + c_{max}/p_{min}$.[5] By Lemma 1 and the above inequlity, $||\boldsymbol{C^N} - \boldsymbol{C}|| \leq (1 - p_{min})||\boldsymbol{C^{N-1}} - \boldsymbol{C}|| \leq \cdots \leq (1 - p_{min})^N ||\boldsymbol{C^0} - \boldsymbol{C}|| \leq (1 - p_{min})^N (\beta_{max} + c_{max}/p_{min}) \leq \epsilon_0$. Taking the logarithm for both sides, we get the inequality in the lemma. □

The following lemma specifies the relationship between the local error bound ϵ and the global error bound ϵ_0.

LEMMA 4. *Let* $\epsilon_0 = \epsilon\,(1 - p_{min})/\,p_{min}$. *If* $||\boldsymbol{C^{k+1}} - \boldsymbol{C^k}|| \leq \epsilon$, *then* $||\boldsymbol{C^{k+1}} - \boldsymbol{C}|| \leq \epsilon_0$.

PROOF SKETCH. This lemma can be proven by applying Lemma 1 and the triangle inequality of max norm. □

Using Lemmas 3 and 4 , we can get the relationship between N and ϵ, and hence time complexity of MEC:

THEOREM 3. *Given the local error bound* ϵ, *the time complexity of MEC is* $O(ndN)$, *where* d *is the maximum degree of a node and* $N = \left(\log \frac{\beta_{max} + c_{max}/p_{min}}{\epsilon(1-p_{min})/p_{min}}\right) / \left(\log \frac{1}{1-p_{min}}\right)$.

Note that, this complexity is derived under the assumption that the network is represented by an adjacency list. Also note that, fixing other parameters such as c_{max} and p_{min}, this complexity is linear in the number of nodes n. Also, taking advantage of being an offline algorithm (i.e. an optimal policy can be computed before the agent starts the actual resource search), if the probabilities for specific time periods do not change very often, then the optimal policies for different time periods (e.g. each hour on each weekday) can be precomputed and stored.

Effect of Penalty $\boldsymbol{\beta}$. Note that in an unbounded search, the optimal solution is not always an infinite path. It is possible that for a certain node v_i, the penalty β_i for terminating the search at v_i is smaller than the minimum expected cost of continuing the unbounded search from v_i (see Equation (2)). In this case, the optimal action for the unbounded search is to terminate the search and take the penalty. For example, consider the taxi-customer search scenario. At 11 pm, for a node v_i that is very close to the driver's home, the driver might consider β_i as a very small quantity, because terminating the customer search at this point does not incur a big loss in terms of income and gas consumption. (Although we assume that the costs, probabilities, and penalties are fixed during a resource search, they can be different for different time intervals, e.g. hours of a day.) However, if β_i is larger than the cost of continuing an unbounded search, in particular if $\beta_i > c_{max}/p_{min}$ (see Lemma 2), then requesting an unbounded search will not produce a path ending at v_i.

Accounting for Observations. Now observe that there are two different variants of the resource-search problem. In one variant the probabilities are unaffected by observations. This implies that every time the agent traverses an edge e, regardless of whether or not the agent traversed e previously, e will have an available resource with the same probability. This assumption represents real-world situations in applications such as taxis searching for customers. However, in applications such as the parking search, since vehicles usually park for some period of time, the probability of a block is not independent of observations. When the agent traverses a block and does not find parking, it is very likely that parking will be unavailable in the next minute, even if the probability of the edge is high.

To capture this intuition, we introduce the notion of the probability *recovery function* [1], which adapts the probabilities according to observations. Specifically, when the agent traverses an edge e and finds no desired resource, the probability of e drops to 0 and monotonically increases to its prior value within the time required for the agent to traverse at most h edges. Then we devise the Adaptive Minimum Expected Cost (AMEC) algorithm, which is an adaptation of MEC to probabilities that behave according to a recovery function. Specifically, using an adaptation of Equations (2), (3), and (4) to probabilities that behave according to the recovery function, AMEC computes the minimum expected cost for each decision point and each finite sequence of edges that may lead the agent to that decision point. The complexity of AMEC for bounded searches is $O(nd^{h+1}K)$.

[5] It can be verified that for max norm, the triangle inequality $||\boldsymbol{U} + \boldsymbol{V}|| \leq ||\boldsymbol{U}|| + ||\boldsymbol{V}||$ holds.

6. REFERENCES

[1] Q. Guo, O. Wolfson, and D. Ayala. A framework on spatio-temporal resource search. In *Proc. 11th IEEE Int. Wireless Communications & Mobile Computing Conf. (IWCMC)*, August 2015.

[2] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *Proc. 29th IEEE Int. Conf. on Data Eng. (ICDE)*, 2013.

[3] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2003.

[4] B. Xu, O. Wolfson, J. Yang, L. Stenneth, and P. S. Yu. Real time street parking availability estimation. In *Proc. 14th Int. Conf. on Mobile Data Management (MDM)*, Milan, Italy, June 3-6, 2013.

[5] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: A recommender system for finding passengers and vacant taxis. *IEEE Trans. Knowl. Data Eng.*, 2013.