Geoinformatica
DOI 10.1007/s10707-016-0275-9

Probabilistic spatio-temporal resource search

Qing Guo[1] · **Ouri Wolfson**[1]

Received: 1 March 2016 / Revised: 18 August 2016 / Accepted: 12 October 2016

Abstract In this paper, we deal with the resource search problem in a probabilistic setting. In a resource search problem, there are spatially located static resources and a mobile agent. The agent looks to obtain one of the resources while minimizing the cost. This cost may consist of different types of costs the agent has to pay, from travel time to the cost of obtaining a certain resource. We assume that the agent has no knowledge of exact availability of the resources in real-time, but some prior or partial data gives estimations of this information. This model applies to many situations that arise in urban transportation systems, such as drivers looking for street parking, taxis looking for new customers, and electric vehicles looking for charging stations. Our approach to the resource search problem only employs uncertain information about resource availability, minimizes the expected cost, and utilizes concepts from decision theory. A simulation that uses real-world data is used to compare our approach to alternatives.

Keywords Spatial databases · Spatio-temporal databases · Probability · Probabilistic data · Routing

1 Introduction

Despite the rapid growth in popularity of mobile devices, wireless embedded sensors, and location-based services, locating geographically-distributed resources still remains

✉ Qing Guo
qguo@cs.uic.edu

Ouri Wolfson
wolfson@cs.uic.edu

[1] Department of Computer Science, University of Illinois at Chicago, Chicago, IL, USA

Published online: 25 November 2016

challenging. Certainly, with the help of smartphones, a taxi driver is able to locate potential passengers [14]; with the help of wireless embedded sensors, a vehicle is able to locate available on-street parking spaces (e.g., the SFpark project[1]). However, not all taxi passengers have smartphones and have subscriptions to taxi request services such as Uber, and even for cities like San Francisco, the cost of installing and maintaining embedded sensors for the whole city is so prohibitive that SFpark only covers a very small portion of the city.

To solve the difficulty of obtaining real-time and accurate resource availability data for resource-search, it is common to use crowdsourcing methods. However, in crowdsourcing, only part of the agents are used for data collection. This often leaves the exact number of available resources per location unknown to an agent. In other words, oftentimes, only uncertain data is available. Therefore, in this work we focus on uncertain data for resource search.

In our problem setup, we assume that there is a directed graph representing a road network, and there are resources located on the edges of the graph. In contrast to Points-of-Interest, a resource may be used by only one agent at a time. An available resource may become unavailable at a later time, and vice versa. Each edge is associated with a cost. This cost may be the cost of traversing the edge (*travel cost*, e.g. travel time), or the travel cost plus the cost of obtaining a resource on that edge (*usage cost*, e.g. walking distance from a parking block to the final destination). We call this combined cost the *general cost*. Each edge is also associated with a probability distribution of the number of available resources during a certain time period. We call any data type that represents the probability distribution an *uncertainty metric*. In this paper, we deal with two uncertainty metrics:[2] one is the probability of having at least one available resource on an edge, and the other is the mean and variance of the number of available resources on an edge.

Using uncertain data, we address the problem of guiding the agent through the road network to efficiently find a desired resource. In other words, we aim to *find an optimal search-path,*[3] *and to do so by an efficient algorithm.*

What is the meaning of an optimal search-path for a resource? An existing way of defining such a path is the Probability Maximization (PM) algorithm [11, 24]. The PM algorithm chooses a path with the maximum overall probability of finding a resource from all paths of length (cost) K or less.

This approach has several drawbacks. First, every infinite path has a probability 1 of finding a resource.[4] Thus, all unbounded search-paths have identical probability. Therefore, PM 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). Second, consider the example in Fig. 1. This road network consists of two nodes and two parallel edges. Each edge has the same probability 0.5. It takes 15 minutes to traverse the upper edge, and 5 minutes to traverse

[1] http://sfpark.org/

[2] The uncertain data may be obtained from historical data, partial real-time data, or a combination of both. One example is Xu et al. [22], where the authors used a crowdsourcing approach to approximate the parking availability per block. Another example is Mathur et al. [15], where the authors proposed using existing municipal vehicles equipped with GPS receivers and ultrasonic sensors to detect on-street parking availability.

[3] Observe that the paths do not have to be Hamiltonian. In other words, an edge is allowed to be visited multiple times, because an unavailable edge (i.e. an edge that does not have an available resource) may later become available.

[4] 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.

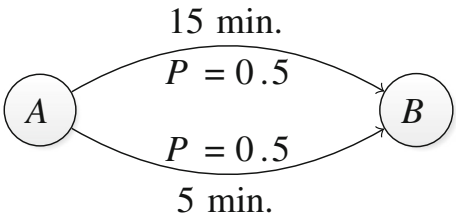


Fig. 1 An example network. The *upper* and *lower edges* have the same probability (0.5), but different travel costs (15 minutes and 5 minutes)

the lower one. Given the maximum search time of 15 minutes, PM may choose either edge because they have the same probability. But the lower path is clearly superior because it is shorter. This difference is not captured by PM. Third, there is no straightforward way to consider usage costs in PM.

Therefore, in this paper we focus on an alternative method, called the General Cost Minimization (GCM), to efficiently search for a resource. As the name suggests, GCM produces a path of minimum expected general cost, and does so efficiently. Consider again the example in Fig. 1. Because the shorter edge has a lower expected cost, it is preferred by GCM. GCM produces such a path efficiently, for bounded and unbounded (i.e. infinite) paths.

The model we develop in this paper applies principles of decision theory, more specifically, the principle of maximum expected utility (in our case, minimum expected general cost). This principle has a long history of development in the artificial intelligence community [18]. According to decision theory, if an agent's utility (e.g. most monetary gains, or least cost) is specified, then an action that optimizes the expected utility will result in the most satisfactory situation. Specifically, we use a dynamic programming approach to compute the minimum expected general cost of a path within a certain length K, as well as for unbounded searches.

Now observe that there are two 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 the agent has traversed e previously, e will have an available resource with the same probability. However, in some applications such as the parking search, since vehicles usually park for some period of time, the availability probability of a block is not independent of observations. When the agent traverses an edge e and does not find parking, it is very likely that parking will be unavailable in the next minute, even if the probability of e is high. To capture this intuition, we apply the notion of the *recovery function*, which adapts the probabilities according to observations [9]. Then we devise the Adaptive General Cost Minimization (AGCM) algorithm, which is a variant of GCM that minimizes the expected general costs for probabilities that behave according to a recovery function.

Finally, we test our algorithms experimentally using real-world data from SFpark, and from GPS taxi trajectories collected in Beijing. We compare our approach with the Probability Maximization (PM) algorithm. The experimental results show that our approach provides significant improvements over PM. And these results hold even in the face of errors in probabilities or means.

Following is a summary of the main contributions of this work:

– We provide an efficient dynamic programming algorithm that finds an optimal path with the minimum expected general cost, for both bounded and unbounded searches. This cost may include not only the cost of traveling the edges, but also other types of costs in general.

- To take advantage of observations made during a search, we adopt the notion of probability recovery function, and devise the corresponding algorithm (AGCM) to compute a path minimizing the expected general cost.
- We adapt our approach that uses the probability as the uncertainty metric to accommodate another uncertainty metric, i.e. the mean and variance.
- We conduct experiments using real-world data from SFpark, and from GPS taxi trajectories collected in Beijing, comparing the performance of our algorithms with PM.
- We evaluate the algorithms' sensitivity to errors in the uncertainty data.

The rest of the paper is organized as follows. Section 2 is a survey of related work. Section 3 presents the basic model of the resource search problem with general costs, and devises the General Cost Minimization algorithm. Section 4 extends the basic model to the Adaptive GCM algorithm by considering the recovery function. Section 5 extends the basic model to unbounded resource-searches and presents the Unbounded GCM algorithm. Section 6 shows how to use the mean and variance as another uncertainty metric. Section 7 presents experimental results. Section 8 concludes the paper.

2 Related work

With the fast increase in the accessibility of location-based services, new approaches for sensing and monitoring available spatial objects have been developed [12, 21]. These detection technologies have encouraged research of urban transportation systems, using either real-time data about resource availability [14], or historical data as prior knowledge [23, 24].

Among these applications, parking detection and search problems have been extensively studied. Vehicles equipped with GPS and ultrasonic sensors are used to generate a map of parking availability [15]. Wireless sensors are used to track open parking spaces in parking facilities [17]. These techniques rely on investing new detection devices. By contrast, more flexible detection techniques that utilize existing end-user smartphones for crowdsourcing were developed. Mobility patterns of smartphone users are used by Ma et al. [13] and Xu et al. [22] to detect parking availability of street blocks, by classifying parking and unparking activities. Another approach using Wi-Fi signature matching was introduced by Nawaz et al. [16] for the detection of unparking activities. Note that crowdsourcing approaches usually generate uncertain data, since they can only penetrate to a portion of the agents involved in the activities affecting the environment.

The parking search problem has been tackled assuming that complete deterministic data is available. For example, reservation systems for parking spaces were studied by several authors [4–6]. Parking by reservation attempts to circumvent the competition for parking. However, existing parking systems are not reservation-based but inherently competitive. Ayala et al. [1–3] introduced parking slot assignment games to analyze parking activities in competitive settings. These systems assume complete deterministic parking availability information, as SFpark is providing. As argued, such information is unlikely to be available on a large scale due to the costs involved in installing and maintaining the sensors.

Using crowdsourcing, resource availability information is often represented in probabilistic settings [10, 11, 19, 20, 24]. Yuan et al. [24] propose to maximize the overall probability of finding a resource within a given time limit. One disadvantage of this

proposal is that it only guarantees that the travel cost of a search stays within a limit, but does not minimize the general cost. By contrast, our model minimizes not only travel costs, but also other costs representing the user's preferences for certain resources.

In Safra et al. [19], the probabilities are about whether a query is satisfied. For example, the query can be whether a restaurant is good according the user's preference. Therefore, the truth value of each spatial location is constant, and there is no need to revisit a location.

Jossé et al. [11] and Verroios et al. [20] assume that the exact availability of resources is given when a search request is initiated. (In contrast, we only assume probabilistic availability information.) Consequently, they solve a different problem. Then some probability decay function is used, which is a decreasing function of time for the probability that a known available resource stays available. These decay functions are similar to our recovery function, except that they go in opposite directions. Ours increases, whereas theirs decreases.

In Jossé et al. [11], Safra et al. [19], and Verroios et al. [20], variants of the traveling salesman problem are used as the theoretical solution. Since it is NP-hard, various heuristics that approximate the optimal solution are proposed by these authors. These techniques compute Hamiltonian paths for the resource search. One disadvantage of using Hamiltonian paths as the solution is that no resource location can be visited more than once. But an unavailable resource location may become available after the agent visits it for the first time.

Jossé et al. [10] considers a model similar to ours. However, the paper states that it is impossible to minimize the cost and maximize the probability of success at the same time, and proposes two solutions, i.e. minimizing the cost for the paths with probabilities greater than a threshold, or vice versa. By contrast, in this paper we show that the expected cost minimization is an effective way of combining costs and probabilities. Moreover, the exact algorithm proposed in Jossé et al. [10] performs searches by expanding all possible routes, resulting in exponential complexity. To mitigate this complexity, the authors propose some pruning techniques. By contrast, due to the application of dynamic programming, our approach is polynomial in the size of the road network.

This paper combines and extends our previous work [7, 8]. Specifically, Guo and Wolfson [7] deal with unbounded search without usage costs, while Guo and Wolfson [8] deal with bounded search with usage costs. In this paper, we present the case of unbounded search with usage costs. This is a nontrivial combination of the previous two cases. This paper also extends the previous papers by providing detailed experimental evaluation, as well as proofs. In addition, in this paper we present a way to combine historical data and real-time data.

3 Basic model

In this section, we present the assumptions and notation we use in the paper (Section 3.1), and define the expected general cost of a path (Section 3.2). After that, we show how to find a path with minimum expected general cost within a certain length (Section 3.3).

3.1 Problem setup

Let us first define a *road network* as a graph. In this graph, the vertices are the intersections of the roads; the directed edges are the road segments connecting the intersections,

indicating the allowed travel directions. We assume that there are n vertices in the road network, denoted by v_i, where $i = 1, 2, \ldots, n$. Also, let edge e_{ij} be the road segment between vertices v_i and v_j.

We define a *search-path* as a path, not necessarily simple, in the road network that is provided by any resource search algorithm 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.

Now we define the edge costs. Let c_{ij} denote the *cost* of e_{ij}. In general, c_{ij} consists of two types of costs. One is the *travel cost*, denoted as tc_{ij}, which is the traversal time of e_{ij}. The other is the *usage cost*, denoted as uc_{ij}, which is the cost of obtaining a resource on e_{ij}. For example, in parking search, the vehicle is usually parked at a different location than the exact address of the final destination of the user, due to availability. If parked at edge e_{ij}, then its usage cost uc_{ij} is the time to walk from e_{ij} to the final destination of the agent. Therefore, $c_{ij} = tc_{ij} + uc_{ij}$. Note that all costs are nonnegative.

The following explains how to determine the final cost of an actual resource search. The final cost depends on whether a search is successful, i.e. whether the agent finds a desired resource following a search-path. If the search is *successful*, then the final cost is the total travel cost of the edges traversed (i.e. the prefix of the search-path), plus the usage cost of the obtained resource, if any. On the other hand, if the search is *unsuccessful*, assume that it ends at v_i, the last vertex on the search-path. Then the final cost is the total cost of the edges on the path, plus a constant β_i ($\beta_i \geq 0$). This β_i denotes the additional penalty for not finding any resource after terminating the search at v_i. For example, in parking search, β_i is the cost of traveling from v_i to a private garage and parking there. In a taxi-customer search, β_i is the cost for the taxi driver of returning home without finding any customer. Often all the β_i's are identical, i.e. independent of i. In the above examples, this means that the cost of garage parking is identical, regardless of the vertex at which the option is exercised; similarly, the cost of giving up the taxi-customer search is often independent of the vertex at which the decision is made. In this case there is a single penalty cost, β.

Note that without penalty β_i, the resource search problem becomes trivial. Specifically, if $\forall i$, $\beta_i = 0$, the minimum expected general cost of any resource search is zero and is given by a zero-length path.

It is necessary to distinguish between travel costs and usage costs. This is because the two types of costs happen at different times during a resource search, and thus affect the final cost of a search differently. Specifically, the travel cost of an edge is added to the final cost as long as the agent has traversed the edge, while the usage cost of an edge is added to the final cost only if the agent has found an available resource on it and obtained the resource. This difference means that simply adding the usage cost to the travel cost and plugging this sum to any existing algorithm will not produce correct results.

Now we define the probability. Denote by p_{ij} the *probability* of edge e_{ij} being available ($0 < p_{ij} < 1$). Intuitively, p_{ij} is the probability that e_{ij} has at least one resource available during a certain time interval (e.g. between 17:00 and 18:00). Consequently, $1 - p_{ij}$ is the probability that no resource is available on e_{ij}.

In Section 3 we assume that the probabilities are fixed for any given search. In Section 4 we demonstrate that this assumption does not always hold in real-world applications; and we adapt the results of Section 3 accordingly.

3.2 Expected general cost of a search-path

We define the expected general cost of a search-path. In our definition, the expected general cost is defined in a recursive (or stepwise) fashion. Specifically, the expected cost of a search-path is defined by the expected cost of its first edge and the expected cost of the sub-path without the first edge. This recursive definition makes it possible to devise a Bellman equation to compute a path with the minimum expected general cost using dynamic programming, as demonstrated in Section 3.3.

For search-path $\{v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_K\}$, denote its expected general cost as $C_{\{v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_K\}}$, then

$$C_{\{v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_K\}} = \begin{cases} tc_{01} + p_{01} uc_{01} + (1 - p_{01}) C_{\{v_1 \rightarrow \cdots \rightarrow v_K\}}, & \text{if } uc_{01} \leq C_{\{v_1 \rightarrow \cdots \rightarrow v_K\}}. \\ tc_{01} + C_{\{v_1 \rightarrow \cdots \rightarrow v_K\}}, & \text{otherwise.} \end{cases} \tag{1}$$

The first case in Eq. 1 deals with the situation when the usage cost of the first edge e_{01} is not greater than the expected cost of the rest of the path. In this case, after traversing the first edge (the first term tc_{01}), the second term ($p_{01} uc_{01}$) gives the expected cost of finding the resource on the first edge of the path, and ending the search there; and the third term ($(1 - p_{01}) C_{\{v_1 \rightarrow \cdots \rightarrow v_K\}}$) is the expected cost of not finding a resource and continuing the search on the rest of the path. The second case in Eq. 1 is when the usage cost of the first edge e_{01} is greater than the expected cost of the rest of the path. In this case, regardless of whether there exists an available resource on the first edge, the agent continues the search without obtaining any resource on e_{01}. Similarly:

$$C_{\{v_1 \rightarrow \cdots \rightarrow v_K\}} = \begin{cases} tc_{12} + p_{12} uc_{12} + (1 - p_{12}) C_{\{v_2 \rightarrow \cdots \rightarrow v_K\}}, & \text{if } uc_{12} \leq C_{\{v_2 \rightarrow \cdots \rightarrow v_K\}}. \\ tc_{12} + C_{\{v_2 \rightarrow \cdots \rightarrow v_K\}}, & \text{otherwise.} \end{cases}$$

$$\cdots$$

$$C_{\{v_{K-1} \rightarrow v_K\}} = \begin{cases} tc_{K-1,K} + p_{K-1,K}\ uc_{K-1,K} + (1 - p_{K-1,K}) \beta_K, & \text{if } uc_{K-1,K} \leq \beta_K. \\ tc_{K-1,K} + \beta_K, & \text{otherwise.} \end{cases}$$

3.3 Expected general cost minimization

In this subsection, we show how to compute the minimum expected general cost of all paths within length K using dynamic programming. The result is composed of two parts: an optimal path and an auxiliary action indicator sequence. Recall that with usage costs, obtaining the first-found available resource along the optimal path may not be the optimal action. To achieve the minimum expected general cost, it is crucial to make sure to obtain an available resource only if its usage cost is lower than the expected general cost of continuing the search along the remaining optimal path. To do this, we produce an action indicator sequence during the computation. This sequence specifies whether the agent should obtain a resource if one is found available along the optimal path.

Before presenting the detailed computation, we first define some notation. For $k \in \{1, \ldots, K\}$, 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 general cost of a path in the k-step look-ahead from v_i. Assume that v_j is an immediate successor of v_i in the network. Denote by C_{ij}^k the expected general cost of a path in the k-step look-ahead from v_i, for which: 1) the first two vertices are v_i and v_j, and 2) the expected general cost of the remaining sub-path within length $k-1$ is the minimum among all paths within length $k-1$ from v_j. We call the pair (v_i, k) a *decision point*.

The following equations indicate how to compute C_i^k and C_{ij}^k recursively, and produce the optimal path and the action indicator sequence:[5]

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

$$C_{ij}^k = \begin{cases} tc_{ij} + p_{ij} uc_{ij} + (1 - p_{ij}) C_j^{k-1}, & \text{if } uc_{ij} \le C_j^{k-1}. \\ tc_{ij} + C_j^{k-1}, & \text{otherwise.} \end{cases} \tag{3}$$

$$next_i^k = \begin{cases} \arg\min\limits_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ C_{ij}^k \right\}, & \text{if } C_i^k \neq \beta_i. \\ -1, & \text{if } C_i^k = \beta_i. \end{cases} \tag{4}$$

$$skip_i^k = \begin{cases} \text{false}, & \text{if } j_0 = next_i^k \neq -1 \text{ and } uc_{ij_0} \le C_{j_0}^{k-1}. \\ \text{true}, & \text{if } j_0 = next_i^k \neq -1 \text{ and } uc_{ij_0} > C_{j_0}^{k-1}. \end{cases} \tag{5}$$

Now we give detailed explanations of the above equations:

Equation 2: When the agent is at decision point (v_i, k), the next node v_j should be chosen such that the expected cost of the remaining path is the minimum. If the penalty β_i for terminating the search is smaller than the minimum expected general cost of the remaining path, or no further search is allowed ($k = 0$), then the agent should stop the search and take penalty β_i.

Equation 3: This computes the expected cost for decision point (v_i, k), given that v_i and v_j are the first two nodes, and assuming that the sub-path from v_j with $(k-1)$-step look-ahead has the minimum expected general cost. This is computation is based on Eq. 1.

Equation 4: At decision point (v_i, k), record by $next_i^k$ the index of the successor node v_j of v_i that minimizes the future expected cost. Let $next_i^k = -1$ to indicate that the search is terminated. We call $next_i^k$ the *optimal action* for the agent at decision point (v_i, k). This is because when the agent is at node v_i and has k more edges to search, $next_i^k$ is the index of the successor node that the agent should proceed to in order to minimize the future expected cost.

Equation 5: $skip_i^k$ is a Boolean value that assists the agent when conducting the resource search. Specifically, at decision point (v_i, k), if $skip_i^k = \text{true}$, then the agent should proceed to node $j = next_i^k$ and continue the resource search, even if edge e_{ij} has an available resource. This is because $skip_i^k$ being true means that the usage cost of e_{ij} is greater than the expected general cost of the rest of the optimal path. On the other hand, if $skip_i^k = \text{false}$, then the agent should obtain a resource from e_{ij} if there is one available.

[5] "s.t." is short for "such that".

Equations 4 and 5 are used to produce the optimal path and the action indicator sequence as follows. Assume that v_0 is the starting node, then $l_1 = next_0^K, l_2 = next_{l_1}^{K-1}, \ldots$, until the first k_0 such that $next_{l_{k_0}}^{K-k_0} = -1$. The resulting path $\{v_0 \rightarrow v_{l_1} \rightarrow \cdots \rightarrow v_{l_{k_0}}\}$ is the optimal path that always minimizes future expected general costs as the agent traverses along this path.

The following theorem confirms the optimality of this approach:

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

Proof See Appendix. □

Equations 2 and 3 can be regarded as the Bellman equation [18] in a dynamic programming method for computing the minimum expected general cost of any path with length K or less. This is because each C_i^k only needs to be computed once.

Algorithm 1 (GCM) computes the minimum expected general costs and the corresponding optimal paths and action indicator sequences. Overall, it builds the minimum expected general costs bottom-up by Eqs. 2 and 3 using dynamic programming for increasing k, until the K is reached. Clearly, Algorithm 1 has a time complexity of $O(ndK)$.

Algorithm 1 General Cost Minimization Algorithm (GCM)

Input: Network $\langle E, V\rangle$, $\{tc_{ij}\}$, $\{uc_{ij}\}$, $\{\beta_i\}$, $\{p_{ij}\}$
Output: $\{C_i^k\}$, $\{next_i^k\}$, $\{skip_i^k\}$

for $i = 1, 2, \ldots, n$ **do**
  $C_i^0 \leftarrow \beta_i$
end for
for $k = 1, 2, \ldots, K$ **do**
  for $i = 1, 2, \ldots, n$ **do**
    $C_i^k \leftarrow \infty$
    for all v_j s.t. e_{ij} exists **do**
      if $uc_{ij} \leq C_j^{k-1}$ **then**
        $C_{ij}^k \leftarrow tc_{ij} + p_{ij}uc_{ij} + (1 - p_{ij})C_j^{k-1}$
        $skip \leftarrow$ false
      else
        $C_{ij}^k \leftarrow tc_{ij} + C_j^{k-1}$
        $skip \leftarrow$ true
      end if
      if $C_{ij}^k < C_i^k$ **then**
        $C_i^k \leftarrow C_{ij}^k$
        $next_i^k \leftarrow j$
        $skip_i^k \leftarrow skip$
      end if
    end for
    if $\beta_i < C_i^k$ **then**
      $C_i^k \leftarrow \beta_i$
      $next_i^k \leftarrow -1$
    end if
  end for
end for

4 Adaptive probabilities

In this section, we present a modified version of the basic resource search model in Section 3 that allows adaptive probabilities. Specifically, in the model of Section 3, the probabilities are assumed to be constant for a given time interval. This implies that every time the agent traverses an edge e, regardless of whether the agent traversed e previously, e will have an available resource with the same probability. This assumption represents real-world situations in some applications. However, in applications such as the parking search, since vehicles usually park for some period of time, the availability probability of a block is not independent of observations. When the agent traverses a parking block and does not find parking, it is very likely that parking will be unavailable in the next minute, even if the probability of this block is high. The model in this section is adapted to take observations into consideration: the observation that there is no availability at the moment is a good indicator that this block will have a lower probability of availability for some time in the future. Therefore, in this section, we allow the probability of an edge to decrease after observing that it is unavailable.

4.1 Probability recovery function

In order to utilize the observations during a search, we define the *recovery function* for the probabilities as follows [9]:

Definition 1 A *recovery function* $p'_{ij} = \text{Recovery}(p_{ij}, t_{ij})$ for the probability of edge e_{ij}, where t_{ij} is the time that has elapsed since edge e_{ij}'s last traversal, is a function of time that is valued at zero when the agent traverses e_{ij} and finds no available resource, and monotonically increases until either of these events occurs: 1) the prior probability p_{ij} is reached, or 2) a new traversal of e_{ij} occurs. In the first case the probability stays at p_{ij}, and in the second it drops back to zero.

4.2 Expected cost minimization with recovery function

In this subsection, we describe a resource search algorithm, derived from GCM, that incorporates the probability recovery function. We call this modified algorithm the *Adaptive General Cost Minimization* (AGCM) algorithm.

Now we make an important assumption on the probability recovery function that we use for AGCM. We assume that the recovery function recovers a probability to its prior value within in a limited time t_0. That is, $\forall t_{ij} \geq t_0, p'_{ij} = \text{Recovery}(p_{ij}, t_{ij}) = p_{ij}$. With this restriction, only a limited history of visited edges needs to be tracked to compute the recovered probabilities. In other words, it prevents the need of tracking a complete traversal history. Denote by h the minimum number of edges needed to be tracked, then $h = \lceil t_0 / c_{min} \rceil$, where $c_{min} = \min_{\text{all } e_{ij}} c_{ij}$.

Now we show how to find the minimum expected general cost of paths with length at most K considering the recovery function. In order to keep track of a history of the last h edges that the agent has considered, we expand the concept of decision point so it contains a history of previously traversed nodes. Specifically, decision point $(v_i | \{v_{i-h}, \ldots, v_{i-1}\}, k)$ means that the agent is currently at node v_i in k-step look-ahead, and that $\{v_{i-h}, \ldots, v_{i-1}\}$ is the previous h nodes the agent has traversed before v_i. Similarly to the basic model, denote by $C^k_{i|i-h,\ldots,i-1}$ the minimum expected general cost for

decision point $(v_i|\{v_{i-h},\ldots,v_{i-1}\},k)$. Denote by $C^k_{i,j|i-h,\ldots,i-1}$ the expected general cost of the paths for decision point $(v_i|\{v_{i-h},\ldots,v_{i-1}\},k)$, such that 1) v_i and v_j are the first two nodes of the path, and 2) the remaining sub-path starting from v_j that is consistent with the history, has a minimum expected general cost with $(k-1)$-step look-ahead. Then similar to Eqs. 2, 3, 4, and 5, these two costs can be computed as:

$$C^k_{i|i-h,\ldots,i-1} = \begin{cases} \min\limits_{\forall j,\text{ s.t. } e_{ij} \text{ exists}} \left\{C^k_{i,j|i-h,\ldots,i-1}, \beta_i\right\}, & \text{if } k > 0. \\ \beta_i, & \text{if } k = 0. \end{cases} \tag{6}$$

$$C^k_{i,j|i-h,\ldots,i-1} = \begin{cases} tc_{ij} + p'_{ij}uc_{ij} + (1-p'_{ij})C^{k-1}_{j|i-h+1,\ldots,i}, & \text{if } uc_{ij} \le C^{k-1}_{j|i-h+1,\ldots,i}. \\ tc_{ij} + C^{k-1}_{j|i-h+1,\ldots,i}, & \text{otherwise.} \end{cases} \tag{7}$$

$$p'_{ij} = \text{Recovery}(p_{ij}, t_{ij}). \tag{8}$$

$$next^k_{i|i-h,\ldots,i-1} = \begin{cases} \arg\min\limits_{\forall j,\text{ s.t. } e_{ij} \text{ exists}} \left\{C^k_{i,j|i-h,\ldots,i-1}\right\}, \\ \qquad\qquad \text{if } C^k_{i|i-h,\ldots,i-1} \ne \beta_i. \\ -1, \qquad\qquad \text{if } C^k_{i|i-h,\ldots,i-1} = \beta_i. \end{cases} \tag{9}$$

$$skip^k_{i|i-h,\ldots,i-1} = \begin{cases} \text{false, if } j_0 = next^k_{i|i-h,\ldots,i-1} \ne -1 \\ \qquad \text{and } uc_{ij_0} \le C^{k-1}_{j_0|i-h+1,\ldots,i}. \\ \text{true, if } j_0 = next^k_{i|i-h,\ldots,i-1} \ne -1 \\ \qquad \text{and } uc_{ij_0} > C^{k-1}_{j_0|i-h+1,\ldots,i}. \end{cases} \tag{10}$$

Note that in the beginning of a search, the length of history may be smaller than h edges. In this case, h in above equations should be replaced by the actual length of history.

Similarly to the basic model in Section 3, based on these equations, a dynamic programming algorithm (i.e. AGCM) can be developed. Specifically, $C^k_{i|i-h,\ldots,i-1}$ is computed by $C^{k-1}_{j|i-h+1,\ldots,i-1,i}$ of every successor node v_j of v_i. This requires the computation of the minimum expected general cost for each decision point. For node v_i, there are at most d^h possible predecessor paths, where d is the maximum degree of a node in the graph. Therefore, for a (v_i, k) pair, there are at most d^h decision points.

Considering the total number of decision points, the time complexity of AGCM is $O(nd^{h+1}K)$. If the length bound of the recovery function h is constant, this time complexity is polynomial in the size (n nodes) of the graph. Although it is exponential in h, we show in Section 7.2 that only a short history for the recovery function is needed to achieve satisfactory performance for real-life problems. In addition, as in the basic model, given the prior probabilities, these optimal paths can be precomputed for each possible starting location and stored for later use. Therefore, for a reasonable h, AGCM is scalable in terms of the network size.

5 Unbounded search optimization

In this section, we expand the basic model in Section 3 to unbounded searches. And then we provide an efficient algorithm to compute the minimum expected general cost of an unbounded search, and find the path of such a search.

First, we make some important assumptions on the usage costs for this section. Note that, if all usage costs for the edges are greater than or equal to their corresponding penalty, i.e. $\forall i, j$ such that e_{ij} exists, it holds that $\beta_i \le uc_{ij}$, then the optimal path is a zero-length path.

In other words, the optimal action for the agent is to not initiate the resource-search and take the penalty right away. Therefore, in this section, we assume that $uc_{ij} \leq \beta_i$. Another assumption is that the usage costs are symmetric, i.e. $\forall i, j$ such that e_{ij} exists, $uc_{ij} = uc_{ji}$. In most applications, such as parking search and taxi-customer search, usage costs are indeed symmetric and lower than the penalties. Based on these assumptions, we present a theorem that simplifies Eq. 3 and hence the search process for resources:

Theorem 2 *If* $\forall i, j$ *such that* e_{ij} *exists,* $uc_{ij} = uc_{ji} \leq \beta_i$, *then* $\forall k$, $uc_{ij} \leq C_j^k$.[6]

Proof See Appendix. □

Theorem 2 means that for symmetric usage costs, if they are not greater than the corresponding penalties, then the second case in Eqs. 3 and 1 will never happen.[7] In other words, there is no need to compare the usage cost with the expected cost of continuing the search; once a resource is found available for the first time during the search, the agent can just obtain it and conclude the search. In this case, Eq. 5 can be disregarded, and Eq. 3 can be rewritten as:

$$C_{ij}^k = tc_{ij} + p_{ij}\, uc_{ij} + (1 - p_{ij})C_j^{k-1}. \tag{3'}$$

Now we present the proof of convergence for this modified problem defined by the assumption that $\forall i, j$ such that e_{ij} exists, $uc_{ij} = uc_{ji} \leq \beta_i$, and Eqs. 2, 3′, and 4.

The convergence proof of the minimum expected general cost of an unbounded search is based on the concept of *contraction* [18], defined as follows. Assume that $\{\boldsymbol{U_k} : k = 1, 2, \dots\}$ is a set of vectors, and $\boldsymbol{B}$ is an operator (e.g. function). Assume that $\boldsymbol{B}$ defines a sequence $\boldsymbol{U_{k+1}} = \boldsymbol{B}\boldsymbol{U_k}$. If there exists $0 \leq \gamma < 1$, such that $\forall k, k' \in \{0, 1, \dots\}$, $||\boldsymbol{U_{k+1}} - \boldsymbol{U_{k'+1}}|| = ||\boldsymbol{B}\boldsymbol{U_k} - \boldsymbol{B}\boldsymbol{U_{k'}}|| \leq \gamma||\boldsymbol{U_k} - \boldsymbol{U_{k'}}||$, then $\boldsymbol{B}$ is a *contraction*.[8] An important property of a contraction is that it has a single fixed point, and repeatedly applying the contraction will lead to that fixed point in the limit (*Banach fixed-point theorem* or *contraction mapping theorem*) [18]. In the resource search problem, consider the vector $\boldsymbol{C^k} = \{C_1^k, \dots, C_n^k\}$ of the minimum expected general costs as the vector $\boldsymbol{U_k}$, and Eqs. 2 and 3′ as the operator $\boldsymbol{B}$. Then the following Lemma 1 indicates that $\boldsymbol{B}$ is a contraction, and the resulting Theorem 3 indicates that the minimum expected general cost of an unbounded search converges:

Lemma 1 *In Eqs. 2 and 3′,* $\forall k, k' \in \{0, 1, \dots\}$, $||\boldsymbol{C^{k+1}} - \boldsymbol{C^{k'+1}}|| \leq \gamma||\boldsymbol{C^k} - \boldsymbol{C^{k'}}||$, *where* $\gamma = 1 - p_{min}$.

Proof See Appendix. □

Theorem 3 *The minimum expected general costs* $\boldsymbol{C^k}$ *defined by Eqs. 2 and 3′ converge to a fixed point when* $k \to \infty$.

Algorithm 2 (UGCM) computes the minimum expected general cost and optimal path of unbounded searches. It builds the minimum expected general costs bottom-up by Eqs. 2, 3′,

[6] As a special case, Theorem 2 holds when all usage costs are zero ($uc_{ij} = 0$ for all edges). In the case of parking this means that the walking time is negligible compared to driving time, which is certainly true when the objective function to minimize is, for example, pollution.

[7] The proof of Theorem 2 applies to Eq. 1 as well.

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

and 4 using dynamic programming for increasing k, until the error is bounded, i.e., k is large enough so that 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.

Algorithm 2 Unbounded General Cost Minimization Algorithm (UGCM)

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

for $i = 1, 2, \ldots, n$ **do**
  $C_i^0 \leftarrow \beta_i$
end for
$error \leftarrow 1 + \epsilon$
$k \leftarrow 1$
while $error > \epsilon$ **do**
  for $i = 1, 2, \ldots, n$ **do**
    $C_i^k \leftarrow \infty$
    for all v_j s.t. e_{ij} exists **do**
      $C_{ij}^k \leftarrow tc_{ij} + p_{ij}\ uc_{ij} + (1 - p_{ij})C_j^{k-1}$
      if $C_{ij}^k < C_i^k$ **then**
        $C_i^k \leftarrow C_{ij}^k$
        $next_i^k \leftarrow j$
      end if
    end for
    if $\beta_i < C_i^k$ **then**
      $C_i^k \leftarrow \beta_i$
      $next_i^k \leftarrow -1$
    end if
  end for
  $error \leftarrow ||\boldsymbol{C}^k - \boldsymbol{C}^{k-1}||$
  $k \leftarrow k + 1$
end while

Infinite path representation Observe that UGCM computes the minimum expected general 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, the same action is recommended, i.e. proceed to the same successor, or terminate the search. Intuitively, whenever the agent arrives at this node, all the unbounded paths are considered; 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*.

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.

In essence, Algorithm 2 is a finite algorithm approximating the optimal policy in an unbounded search. Therefore, for large enough $k = k_0$ such that the error is bounded by ϵ, the optimal policy in theory is estimated by $next_i = next_i^{k_0}$, $i = 1, \ldots, n$.

Complexity of UGCM The complexity of UGCM 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 UGCM; the *global error bound* ϵ_0 measures the distance between the current value of $\boldsymbol{C^k}$ and its converged value (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 *If* $\forall i, j$ *such that* e_{ij} *exists,* $uc_{ij} = uc_{ji} \leq \beta_i$, *then the expected cost of unbounded path* $\{v_0 \to v_1 \to \cdots\}$ *is bounded by the following:*

$$C_{\{v_0 \to v_1 \to \cdots\}} \leq \frac{tc_{max} + uc_{max} \cdot p_{max}}{p_{min}},$$

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

Proof See Appendix. □

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 UGCM. For a given global error bound* ϵ_0, *if the following inequality holds:* $N \geq \left(\log \frac{\beta_{max} + (tc_{max} + uc_{max} \cdot p_{max})/p_{min}}{\epsilon_0}\right) / \left(\log \frac{1}{1-p_{min}}\right)$, *then* $||\boldsymbol{C^N} - \boldsymbol{C}|| \leq \epsilon_0$.

Proof (*Sketch*) By Lemma 2, the maximum initial error is $||\boldsymbol{C^0} - \boldsymbol{C}|| \leq ||\boldsymbol{C^0}|| + ||\boldsymbol{C}|| \leq \beta_{max} + (tc_{max} + uc_{max} \cdot p_{max})/p_{min}$. [9] By Lemma 1 and the above inequality, $||\boldsymbol{C^N} - \boldsymbol{C}|| \leq (1 - p_{min})||\boldsymbol{C^{N-1}} - \boldsymbol{C}|| \leq \cdots \leq (1 - p_{min})^N ||\boldsymbol{C^0} - \boldsymbol{C}|| \leq (1 - p_{min})^N (\beta_{max} + (tc_{max} + uc_{max} \cdot p_{max})/p_{min}) \leq \epsilon_0$. 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* $\epsilon_0 = \epsilon\ (1 - p_{min})\ /\ p_{min}$. $\forall k$, *if* $||\boldsymbol{C^{k+1}} - \boldsymbol{C^k}|| \leq \epsilon$, *then* $||\boldsymbol{C^{k+1}} - \boldsymbol{C}|| \leq \epsilon_0$.

Proof See Appendix. □

When implementing UGCM, one would normally want to guarantee the global error to be bounded by ϵ_0, that is, the error between the minimum expected general costs and their estimated values by UGCM. Given ϵ_0 (for example $\epsilon_0 = 0.1$ minutes), the local error bound ϵ used in Algorithm 2 should be computed based on Lemma 4.

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

Theorem 4 *Given the local error bound* ϵ, *the time complexity of UGCM is* $O(ndN)$, *where* d *is the maximum degree of a node in the network and* $N = \left(\log \frac{\beta_{max} + (tc_{max} + uc_{max} \cdot p_{max})/p_{min}}{\epsilon(1-p_{min})/p_{min}}\right) / \left(\log \frac{1}{1-p_{min}}\right)$.

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

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 tc_{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, its penalty β_i is smaller than the minimum expected cost of continuing the unbounded search from v_i (see Eq. 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 relatively small quantity (still larger than its corresponding usage cost), 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 and penalties stay the same during any resource search, they can be different for different time intervals, e.g. hours of a day.)

On the other hand, if the penalties are large enough, then the agent should never terminate the resource search. Recall that the upper bound on the expected cost for any infinite path is $(tc_{max} + uc_{max} \cdot p_{max}) / p_{min}$ (see Lemma 2). Therefore, if $\beta_i > (tc_{max} + uc_{max} \cdot p_{max}) / p_{min}$ $(\forall i = 1, \ldots, n)$, then for any decision point, terminating the search will not be the optimal action. In other words, the optimal path in this case is an infinite path, and the penalty should never be taken. Moreover, in this case, the optimal policy computed by UGCM is not related to the specific value of β_i. An intuitive explanation is that the probability of not finding a resource following the unbounded optimal path is zero, therefore the penalty for that is negligible.

Observe that following any infinite path, with probability one, the agent will find a desired resource. This is because the probability of finding a resource on any edge is positive, so the accumulated probability that the agent cannot find any resource along the infinite path is zero. Therefore, in applications such as taxi-customer search, if the agent does not have a time constraint to search for a resource, but does care about finding a resource with probability 1, then an unbounded resource search with large penalties will force searching until a resource is found.

6 Mean and variance as uncertainty metric

The uncertainty metric obtained from crowdsourcing is often the mean and variance of the number of available resources per edge, instead of probability [22]. In this section, we present how to convert the mean and variance to probability so our approach can apply to them.

6.1 Convert mean and variance to probability

To convert the mean and variance to probability, we assume that the number of available resources on an edge during a certain time period follows a discretized Gaussian distribution. The reasons to choose the Gaussian distribution are that it only requires the mean and variance to describe, and the conversion between the mean and variance and the

probability is simple and tractable. Empirically, we validated that indeed the estimations of the probability from the mean and variance using Gaussian distribution are very close to the actual probabilities.

Now we present the details. Assume that for an edge e, the number of available resources on e is R ($R \in \{0, \ldots, R_{max}\}$), where R_{max} is the maximum capacity of e. We assume that R is associated with a real-valued random variable X that follows Gaussian distribution. Specifically, $\Pr(R = 0) = \Pr(X < 0.5), \Pr(R = 1) = \Pr(0.5 \leq X < 1.5), \ldots, \Pr(R = R_{max} - 1) = \Pr(R_{max} - 1.5 \leq X < R_{max} - 0.5), \Pr(R = R_{max}) = \Pr(X \geq R_{max} - 0.5)$. Therefore, the probability of edge e being available is $\hat{p} = \Pr(R \geq 1) = \Pr(X \geq 0.5) = \int_{0.5}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$, where μ is the mean and σ^2 is the variance of the number of available resources. Assume there are m samples of the number of available resources on e, denoted by $r_1, \ldots, r_m$. Then the mean μ can be estimated by sample mean $\hat{\mu} = \bar{r} = \frac{1}{m}\sum_{l=1}^{m} r_l$; the variance σ^2 can be estimated by unbiased sample variance $\hat{\sigma}^2 = s^2 = \frac{1}{m-1}\sum_{l=1}^{m}(r_l - \bar{r})^2$. Replacing the probability in our algorithms with above $\hat{p}$, one can use the mean and variance of the number of available resources for each edge to compute the optimal paths. We call the modified algorithms GCM-m, AGCM-m, and UGCM-m.

6.2 Combining historical data and real-time data

To improve the accuracy of estimations in the mean and variance, we use the Kalman filter method introduced by Xu et al. [22] to combine historical data and (partial) real-time data. Unlike other methods, Kalman filter is free of external parameters, hence does not require parameter tuning.

Now we talk about the estimation for a certain block during a certain time period (e.g. between 17:00 and 18:00). Denote the historical mean and variance of its availability as q and Q. Also denote the real-time mean and observational variance as a and A. Then, the estimated mean by Kalman filter is $\hat{\mu} = \frac{A \cdot q + Q \cdot a}{Q + A}$, and the estimated variance is $\hat{\sigma}^2 = \frac{Q \cdot A}{Q + A}$. Note that $\hat{\mu}$ is a weighted average of the historical and real-time mean. Observe that for the two types of data, the one with a larger variance has a smaller weight in $\hat{\mu}$. This confirms the intuition that the more fluctuating data contributes less to the final estimation.

7 Experiments

To test our models, we compare them with a baseline algorithm using real-world data from SFpark, and from GPS taxi trajectories collected in Beijing.

7.1 Baseline algorithm

The baseline algorithm we use is the Probability Maximization (PM) algorithm adapted from Jossé et al. [11] and Yuan et al. [24]. For any specific path $\{v_0 \to v_1 \to \cdots \to v_K\}$, the overall probability of getting at least one available resource along it, is $P_{\{v_0 \to v_1 \to \cdots \to v_K\}} = 1 - \prod_{i=1}^{K}(1 - p_{i-1,i})$. PM chooses a path with the maximum overall probability of finding a resource from all possible paths within a certain length. We use the dynamic programming technique similar to our algorithms to compute these paths.

Now we present the details of PM. Analogous to Eq. 1, the probability of a path and its sub-path without the first edge have relationship $P_{\{v_0 \to v_1 \to \cdots \to v_K\}} = 1 - (1 -$

$p_{01})\left(1 - P_{\{v_1 \to \cdots \to v_K\}}\right)$. Similarly to GCM, we first define some notation for decision point (v_i, k) (i.e. when the agent is at node v_i and has k more edges to search). Denote by P_i^k the maximum probability of a path in the k-step look-ahead from v_i. Assume that v_j is an immediate successor of v_i in the network. Denote by P_{ij}^k the probability of a path in the k-step look-ahead from v_i, for which: 1) the first two vertices are v_i and v_j, and 2) the probability of the remaining sub-path within length $k-1$ is the maximum among all paths within length $k-1$ from v_j. Then analogous to Eqs. 2, 3, and 4, we have[10]

$$P_i^k = \begin{cases} \max\limits_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ P_{ij}^k \right\}, & \text{if } k > 0. \\ 0, & \text{if } k = 0. \end{cases}$$

$$P_{ij}^k = 1 - (1 - p_{ij}) \left(1 - P_j^{k-1}\right).$$

$$next_i^k = \begin{cases} \arg\max\limits_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ P_{ij}^k \right\}, & \text{if } k > 0. \\ -1, & \text{if } k = 0. \end{cases}$$

Using these equations, a dynamic programming algorithm similar to GCM can be implemented. We call it the Probability Maximization (PM).

Similarly to GCM, PM can be adapted to accommodate adaptive probabilities. For example, Jossé et al. [11] assume that when a search request is initiated, some resource locations are known to be available (i.e. their probabilities are one), and then their probabilities decay with time. They propose to pick a Hamiltonian path with the highest overall probability considering the decay. For our parking experiments in Section 7.2, we change the probabilities according to the recovery function.

7.2 Parking search

7.2.1 Simulation setup

The San Francisco Municipal Transportation Agency (SFMTA) initiated a project called SFpark, in which wireless sensors are embedded under the pavement of street parking spaces to detect their occupancies in real-time. Per-block availability was published in real time. Most cities do not have the financial resources that SFpark requires; therefore, the approach developed in our work could benefit urban transportation systems in general. We used the SFpark data which we collected in the simulation environment described next.

A tuple has schema $\langle$*blockId*, *availability*, *operational*, *timestamp*$\rangle$ in SFpark database. It is a report that at time *timestamp*, the number of available parking spaces on block *blockId* changes to *availability*. *operational* is the total number of parking spaces on that block. Given this database, the parking availability of any block at any time can be retrieved. This is done by a query that takes a *blockId* and a time t as input, and searches for the latest report of *blockId* whose *timestamp* is earlier than t.

In our experiments, we use the historical availability reports from SFpark for a tourist area in San Francisco called Fisherman's Wharf. We built a graph of the road network of this area based on the block data given by the SFpark database. The graph consists of 40 nodes

[10]There is no equivalence of Eq. 5 for PM. This is because in PM, the agent should obtain the first resource that is found available.

and 63 edges. This is a reasonable size for street-parking search, because such a search usually involves a small area. Even for a large graph representing for example a city, the parking search algorithm can be performed on a subgraph within a certain radius from the user's destination, e.g. a subgraph in which the walking time from every node to the user's destination is at most 30 minutes. Indeed, in our experiments we limit the maximum path-length of the searches to 30 blocks (edges), considering the average travel cost of an edge in the network is about one minute.

We conducted our simulations for a weekday between 20:00 and 21:00. For each simulation, we randomly and uniformly generate a starting time between 20:00 and 21:00. Similarly, the initial location for each simulation is generated uniformly at random. We assume that the user's final destination is the same as the initial location (i.e. a node in the network). In other words, we assume that the user requests parking navigation when she is at her final destination and has not found parking there. The usage cost of each edge is the shortest walking time from this edge to the user's final destination. To avoid complications that would be irrelevant to the search time, we assume that the vehicle moves at a constant velocity. To test the algorithms for different availability levels (a measure of the scarcity of resources), we artificially removed a certain percentage of the parking spaces on each block in some experiments. For each combination of the parameters, we conducted 10,000 simulations that compute the average final cost for each algorithm.

We compute the uncertainty metrics of each block in the following way, which resembles a process of hourly data collection using a detection vehicle equipped with ultrasonic sensors [15]. We scan the previous 20 weekdays for parking availability data; for each hour on each day, we use a fixed route to scan all the blocks from the beginning of that hour. We record the number of available parking spaces when scanned for each block. And then, for each block, we compute the proportion of days on which the edge is available, i.e. there is at least one available parking space. We use this proportion as the probability of the blocks. The historical mean and variance of a block are estimated by the sample mean and unbiased sample variance of these numbers following Section 6. The real-time mean is estimated by the last scanning before the resource-search request. The real-time variance is estimated by the average hourly squared difference from the beginning of the day of search request. For example, if the availability of a block has been hourly scanned as 3,2,1,0 since 8:00, then the real-time variance for a request at 11:30 is estimated as $\frac{(3-2)^2+(2-1)^2+(1-0)^2}{3} = 1$, and real-time mean is estimated as 0. This estimation is from the assumption that the hourly transition distribution of availability throughout the same day stays the same.

7.2.2 Recovery function for the experiments

There is a wide range of choices for the recovery function defined in Section 4.1. In our experiments we found that as long as the probability stays low for a reasonable time period after traversal, the results are similar, regardless of the specific choice of the recovery function. Therefore, we use a simple step function:

$$p'_{ij} = \text{Recovery}(p_{ij}, t_{ij}) = \begin{cases} 0, & \text{if } t_{ij} < 2 \text{ min.} \\ p_{ij}, & \text{otherwise.} \end{cases} \tag{11}$$

In this equation, t_{ij} is the time that has elapsed since block e_{ij}'s last traversal by the searching agent, and 2 minutes is a threshold. Equation 11 means that the recovery function is a step function which is 0 if the time after the last traversal is smaller than 2 minutes, and p_{ij} otherwise. The threshold of 2 minutes was determined experimentally such that it optimizes the performance of both PM and AGCM. Note that the 2-minute threshold is long enough

to prevent the agent from continuously going back and forth on the same edge. Experimentally we determined that this fixation on an edge is undesirable, and is exactly the reason for introducing the recovery function.

7.2.3 Results

We tested the algorithms for different unavailability levels, i.e. percentages of parking spaces removed. The results are shown in Figs. 2 and 3. In these figures, AGCM-m is the version of ACGM using historical mean and variance of the availability data, and AGCM-KF is the version of ACGM-m using Kalman filter combining historical and real-time data. In Fig. 2, usage costs are not considered; in Fig. 3, usage costs are considered. The results coincide with the intuition: the higher the unavailability level, the longer time on average it takes to find parking (and walk to the final destination) for the algorithms.

As Figs. 2 and 3 indicate, the improvements that AGCM brings over PM are significant. Also, the performance of AGCM-KF is very similar to that of AGCM, suggesting that in parking search, using the mean and variance of the number of available resources to replace the probabilities does not compromise the performance of our approach. Note that, combining historical data with real-time data (AGCM-KF) does improve the performance over using historical data only (AGCM-m) in some cases, but not always. This conclusion is supported by Xu et al.'s work [22], which shows that Kalman Filter improves Boolean accuracy (percentage of correct classifications as available or unavailable) only marginally.

7.3 Taxi-customer search

In this set of experiments we use Beijing's road network as the graph. It has 121,317 nodes and 161,758 edges. We use the Beijing taxi trajectory dataset used by Ma et al. [14] and Yuan et al. [24] to generate customer requests. The trajectories are segmented into individual trips, some of which are occupied trips and others unoccupied. In this dataset, there are

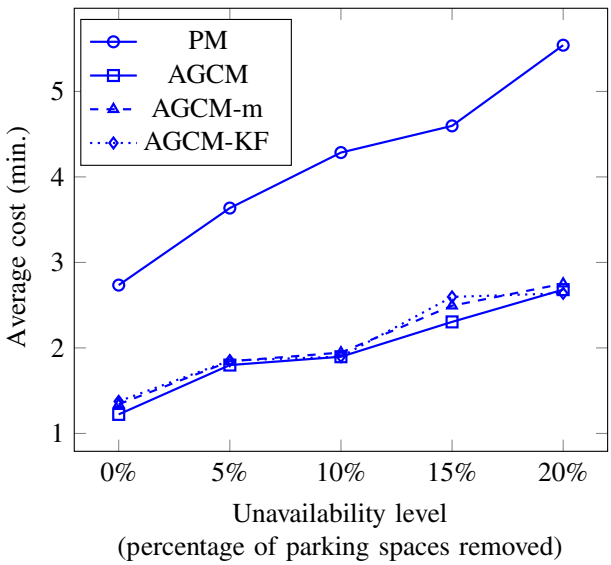


Fig. 2 Average time to find parking with different unavailability levels. Usage costs are zero

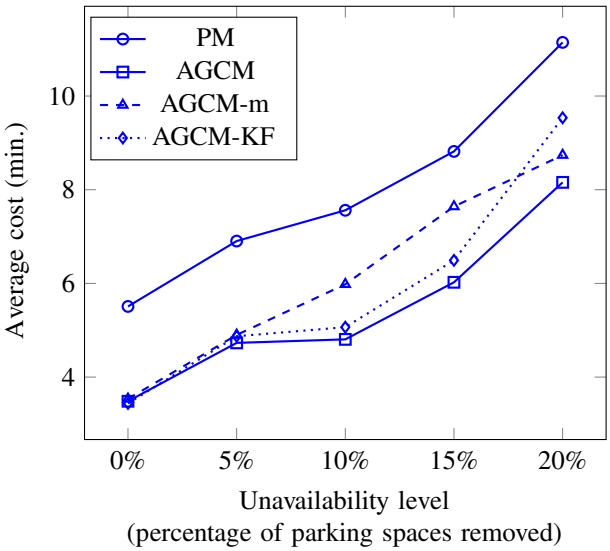


Fig. 3 Average cost for parking with different unavailability levels, with non-zero usage costs

3,787,022 occupied trips on a particular day. We consider the time and location of the origin of each occupied taxi trip as the time and location of a taxi-ride request.

Following Ma et al. [14], we generate the probabilities of the edges in the following way. We divide an hour into 12 time bins, each with a span of 5 minutes. For edge e_{ij}, we count the number of time bins r_{ij} in which there are taxi-ride requests on it during that hour, and estimate the probability of e_{ij} by $p_{ij} = r_{ij}/12$. In the dataset, there are 195,520 taxi ride requests during the hour that we used to compute the probabilities. In order to model unsatisfied requests, and also to test different demand levels for taxi rides, as in Ma et al. [14] we inflate the probabilities for some experiments by a multiplier δ. (If a probability becomes bigger than one after multiplied by δ, we use one.) The mean and variance of an edge is estimated by the sample mean and unbiased sample variance as in Section 6 using the number of taxi ride requests in each time bin on the edge as samples. For a demand level inflated by δ, we compute the sample mean and unbiased sample variance with the inflated samples each multiplied by δ.

Taxi drivers in Beijing usually do not like long trips that end up in remote areas, because those increase the likelihood of having to drive back to the city center empty. This preference needs to be encoded in the usage cost. However, the destinations of passengers from the same starting location are usually different, and they are usually unknown to the taxi driver before a passenger is picked up. Therefore, we approximate the usage costs of the edges by the average trip length for each edge as the trip origin. Specifically, for an edge e, if on average, trips starting from e are longer than 15 km, we assign a usage cost of half of its average trip duration to e. This means that approximately half of the trips that are longer than 15 km end up in remote areas.

We generate the stream of taxi-ride requests for each edge as follows. For edge e_{ij}, during each 30-second interval since the starting time, we generate a taxi-ride request with probability p_{ij}. We assume that a taxi request is only active inside the interval during which it is generated. We conducted 100,000 experiments for each data point in our plots; for each

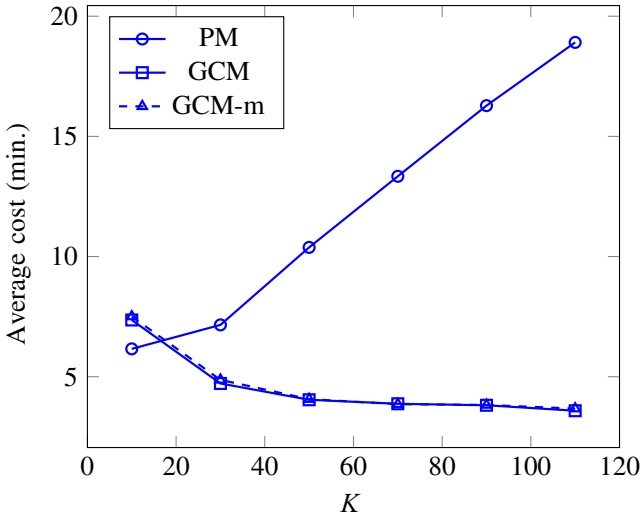


Fig. 4 Average cost for different values of maximum length K, with non-zero usage costs. Demand level $\delta = 1.5$

experiment the initial location of the taxi was chosen randomly, and three search-paths were generated, corresponding to the three algorithms being compared.

We first tested the effect of the length bound K for bounded searches. The result is shown in Fig. 4.[11] It can be seen that for GCM, the performance improves with increasing K. However, the decrease in the resulting average costs is slower for larger K. This means that the performance improvement resulted from an increase in K may disappear when K is sufficiently large. This suggests that for GCM, the minimum expected general costs do converge. By contrast, also from Fig. 4, the performance of PM worsens quickly with increasing K. This is because when $K \to \infty$, almost all paths have identical probability (i.e. probability 1), making different paths indistinguishable.

To verify the convergence of the minimum expected general costs in UGCM, we did another experiment as shown in Fig. 5. In this figure, the horizontal axis is K (i.e. the maximum length of a search), and the vertical axis is $||\boldsymbol{C}^k - \boldsymbol{C}^{k-1}||$, which is marked as *error* in the figure, because it measures how far the minimum expected general costs are from their converged values. Clearly, from Fig. 5, the minimum expected general costs converge because the error approaches to zero as $K \to \infty$. Specifically, for example, when $K = 170$, the error is 2×10^{-4} minutes. This means that when $K \geq 170$, increasing K by one will result in the improvement of the minimum expected general costs by at most 2×10^{-4} minutes.

We also tested PM, UGCM, and UGCM-m for different demand levels (measured by multiplier δ). For PM, we used $K = 30$ as the maximum length of a search, and we make the agent restart the search if no customer is found after traversing 30 edges. The results, as shown in Fig. 6 (where usage costs are zero) and Fig. 7 (with non-zero usage costs), indicate that the more frequent the taxi-ride requests are, the faster to find a customer along

[11] Note that in Fig. 4, the GCM curve is below the GCM-m curve. They appear close due to the larger scale of the vertical axis.

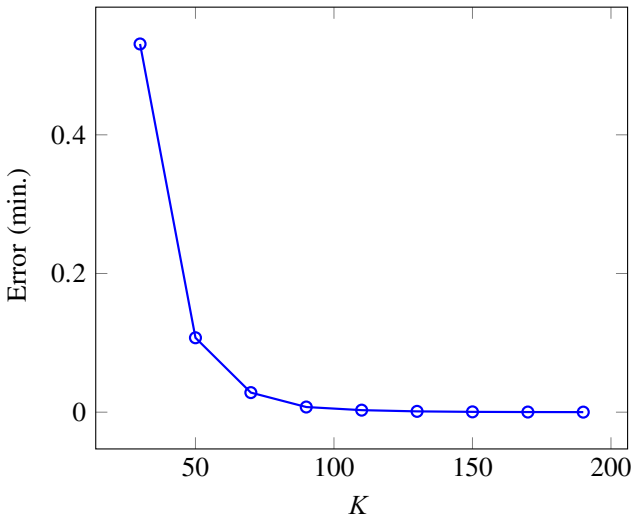


Fig. 5 Error in expected general costs for different values of K, for GCM with non-zero usage costs. Demand level $\delta = 1.5$

a search, for all algorithms. Figures 6 and 7 also show that the performance of UGCM is superior to PM, measured by the average final costs of the resource searches. We also see that the performance of UGCM-m is very close to that of UGCM.

To test how sensitive the algorithms are to data errors, we introduce errors to the probabilities and means in the following way. For a given error rate e_r, with 50 % chance, we add

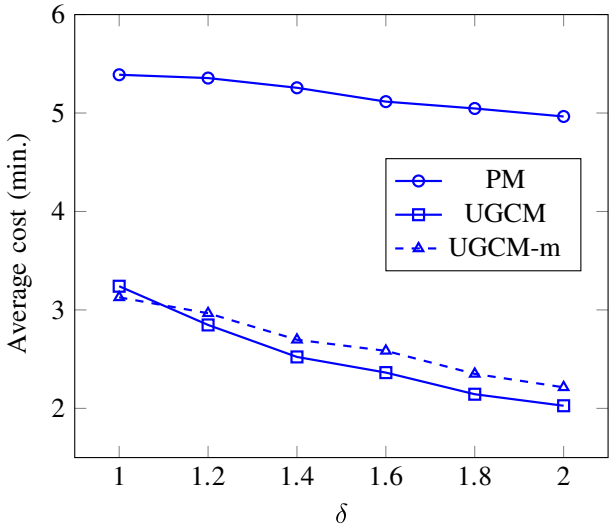


Fig. 6 Average time to find a customer for different demand levels (measured by δ). Usage costs are zero

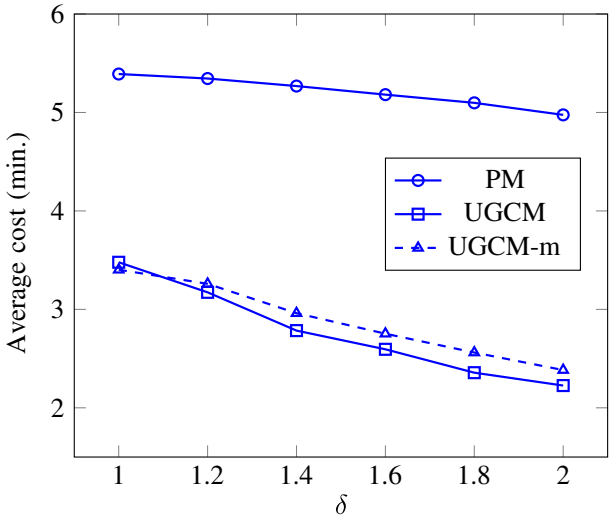


Fig. 7 Average cost for different demand levels (measured by δ), with non-zero usage costs

this percentage to the original data, and with 50 % chance, we substract this percentage from the original data. Specifically, after introducing errors, the probability p of an edge becomes $p(1 + e_r)$ or $p(1 - e_r)$; the sample mean $\bar{r}$ becomes $\bar{r}(1 + e_r)$ or $\bar{r}(1 - e_r)$. The result is shown in Fig. 8. Clearly, UGCM and UGCM-m outperform PM even with data errors.

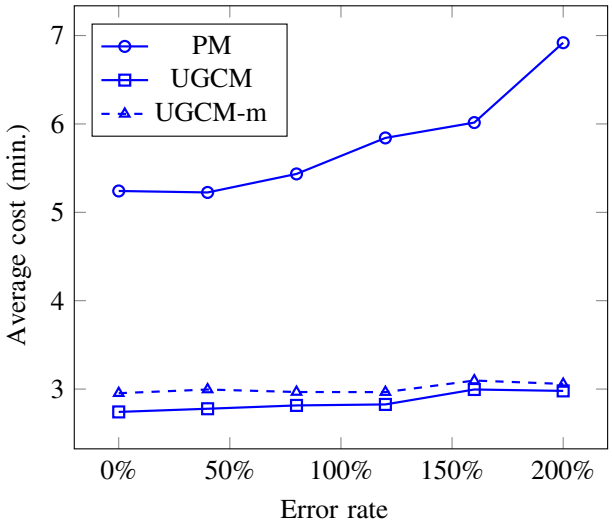


Fig. 8 Average cost for different error rates, with non-zero usage costs. Demand level $\delta = 1.5$

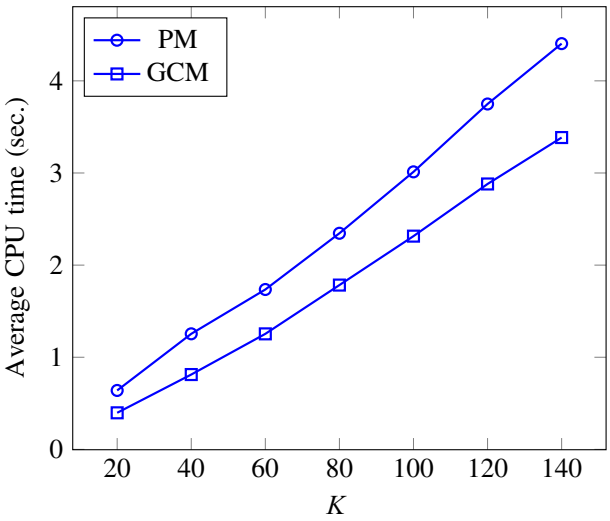


Fig. 9 Average CPU time of running the algorithms for different values of K

Finally, we evaluate the running time of the algorithms. Figure 9 shows the average CPU time of the two algorithms in comparison for different values of K. Clearly, the computation time of GCM is linear in K as shown in Section 3.3.

8 Conclusions

In this paper, we use the minimum expected general cost as the optimization criterion for the spatio-temporal resource search problem with uncertain data. Clearly, it is beneficial to be able to consider general costs that include usage costs in the resource search problem. We show that our optimization criterion is advantageous over other possible optimization criteria such as Probability Maximization.

In applications such as on-street parking search, traversing a resource location and not finding the desired resource means an immediate decrease of its probability. To utilize this observation, we propose the Adaptive General Cost Minimization algorithm. This algorithm has a time complexity that is exponential in the length of the path that needs to be tracked during recovery. However, its efficiency is still reasonable because the length of recovery-path is usually very small. This is demonstrated by experiments. For example, in the SFpark case the probability recovers to its initial value within 2 minutes, thus the algorithm is exponential in the number of edges that can be traversed in 2 minutes.

Under certain conditions, the expected cost of a search is finite even if the search time is unbounded, and the Unbounded General Cost Minimization algorithm finds the tight lower bound, and does so efficiently. Furthermore, it finds the path corresponding to the minimum expected general cost, even if the path is infinite. This fact is useful for applications such as taxi-customer search, because an unbounded search gives a probability of one for finding a resource.

For some data collection methods, the uncertain data about the availability of the resources may be the mean and variance of the number of resources for each location instead

of the probability of having a least one available resource. We use discretized Gaussian distribution to convert the mean and variance to probability. Experiments show that applying the mean and variance to our approach produces comparable results.

Finally, we demonstrate by experiments that, even with data errors, our approach outperforms Probability Maximization.

Acknowledgments This work was supported in part by the NSF under grants IIS-1213013 and IIP-1534138.

Appendix: Proofs

Proof for Theorem 1 First, we define some notation. Let $(C_i^k)'$ denote the expect cost (not necessarily the minimum) of any possible path with k-step look-ahead from node v_i. Let $(C_{ij}^k)'$ denote the expected cost (not necessarily the minimum) of any possible path with k-step look-ahead given that v_i and v_j are the first two nodes of the path.

When $k = 0, \forall i = 1, \ldots, n$, $C_i^0 = \beta_i$ is the only possible expected cost for the zero-length path starting from v_i. Therefore C_i^0 is the minimum.

Assume that when $k = k_0 - 1, \forall i = 1, \ldots, n$, $C_i^{k_0-1}$ is the minimum expected general cost for all paths starting at v_i with length $k_0 - 1$ or less. That is, $C_i^{k_0-1} \leq (C_i^{k_0-1})'$.

Then, when $k = k_0$, for any given v_i, we consider two categories of possible successor nodes for v_i ($\{v_{j'}\}$ and $\{v_{j''}\}$):

1) $\forall j'$ such that $e_{ij'}$ exists and $uc_{ij'} \leq C_{j'}^{k_0-1}$.

By Eq. 3,

$$C_{ij'}^{k_0} = tc_{ij'} + p_{ij'}\, uc_{ij'} + (1 - p_{ij'})C_{j'}^{k_0-1}.$$

Because $uc_{ij'} \leq C_{j'}^{k_0-1} \leq (C_{j'}^{k_0-1})'$, by Eq. 1,

$$(C_{ij'}^{k_0})' = tc_{ij'} + p_{ij'}\, uc_{ij'} + (1 - p_{ij'})(C_{j'}^{k_0-1})'.$$

Clearly, $C_{ij'}^{k_0} \leq (C_{ij'}^{k_0})'$.

2) $\forall j''$ such that $e_{ij''}$ exists and $uc_{ij''} > C_{j''}^{k_0-1}$.

By Eq. 3,

$$C_{ij''}^{k_0} = tc_{ij''} + C_{j''}^{k_0-1}.$$

The relationship between $uc_{ij''}$ and $(C_{j''}^{k_0-1})'$ in this second case can be either way.

a) When $uc_{ij''} \leq (C_{j''}^{k_0-1})'$, by Eq. 1,

$$(C_{ij''}^{k_0})' = tc_{ij''} + p_{ij''}\, uc_{ij''} + (1 - p_{ij'})(C_{j''}^{k_0-1})'.$$

In this case, $C_{ij''}^{k_0} \leq (C_{ij''}^{k_0})'$ because $C_{j''}^{k_0-1} < uc_{ij''}$ and $C_{j''}^{k_0-1} \leq (C_{j''}^{k_0-1})'$.

b) When $uc_{ij''} > (C_{j''}^{k_0-1})'$, by Eq. 1,

$$(C_{ij''}^{k_0})' = tc_{ij''} + (C_{j''}^{k_0-1})'.$$

In this case, again $C_{ij''}^{k_0} \leq (C_{ij''}^{k_0})'$.

Integrating cases 1) and 2), we can conclude that $\forall j$ such that e_{ij} exists, $C_{ij}^{k_0} \leq (C_{ij}^{k_0})'$. Therefore, by Eq. 2, $\forall j$ such that e_{ij} exists, $C_i^{k_0} \leq \min\left\{C_{ij}^{k_0}, \beta_i\right\} \leq \min\left\{(C_{ij}^{k_0})', \beta_i\right\}$. Because $\{v_j\}$ here covers all possible successors of v_i, we can get $C_i^{k_0} \leq (C_i^{k_0})'$.

By induction, we can reach the conclusion that C_i^K is the minimum expected general cost for all paths starting at v_i with length K or less, in other words, $C_i^K \leq (C_i^K)'$.

The time complexity $O(ndK)$ is a direct result of applying Eqs. 2 and 3 for dynamic programming (refer to Algorithm 1). □

Proof for Theorem 2 When $k = 0, \forall i, j$ such that e_{ij} exists, by Eq. 2, $uc_{ij} \leq \beta_j = C_j^0$.

Assume that when $k = k_0 - 1, \forall i, j$ such that e_{ij} exists, it holds that $uc_{ij} \leq C_j^{k_0-1}$, then for $k = k_0$ and any given $i, \forall j$ such that e_{ij} exists, by Eq. 3:

$$\begin{aligned} C_{ij}^{k_0} &= tc_{ij} + p_{ij}\, uc_{ij} + (1 - p_{ij})C_j^{k_0-1} \\ &\geq tc_{ij} + p_{ij}\, uc_{ij} + (1 - p_{ij})uc_{ij} \\ &\geq uc_{ij} \end{aligned}$$

Therefore,

$$\begin{aligned} C_i^{k_0} &= \min_{\forall j,\ \text{s.t. } e_{ij} \text{ exists}} \left\{C_{ij}^{k_0},\ \beta_i\right\} \\ &\geq uc_{ij} \end{aligned}$$

This can be rewritten as $uc_{ij} = uc_{ji} \leq C_j^{k_0}$.

By induction, we can get the conclusion in the theorem. □

The following lemma is necessary to prove Lemma 1:

Lemma 5 *Let* $\{a_i\}, \{b_i\}, i = 1, 2, \ldots, n$ *be two real sequences with the same length n, then*

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

Proof

$$\begin{aligned} \min_{1\leq i\leq n}\{a_i\} - \min_{1\leq i\leq n}\{b_i\} &= \max_{1\leq i\leq n}\left\{\min_{1\leq j\leq n}\{a_j\} - b_i\right\} \\ &\leq \max_{1\leq i\leq n}\{a_i - b_i\} \\ &\leq \max_{1\leq i\leq n}\{|a_i - b_i|\}. \end{aligned}$$

$$\begin{aligned} \min_{1\leq i\leq n}\{a_i\} - \min_{1\leq i\leq n}\{b_i\} &= \min_{1\leq i\leq n}\left\{a_i - \min_{1\leq j\leq n}\{b_j\}\right\} \\ &\geq \min_{1\leq i\leq n}\{a_i - b_i\} \\ &= -\max_{1\leq i\leq n}\{-(a_i - b_i)\} \\ &\geq -\max_{1\leq i\leq n}\{|a_i - b_i|\}. \end{aligned}$$

From these, we can get the inequality in the lemma. □

Proof for Lemma 1 Firstly, we prove that $\forall 1 \leq i \leq n, |C_i^{k+1} - C_i^{k'+1}| \leq \gamma ||\boldsymbol{C^k} - \boldsymbol{C^{k'}}||$ using Lemma 5 and Eqs. 2 and 3′:

$$
\begin{aligned}
&|C_i^{k+1} - C_i^{k'+1}| \\
&= \left| \min_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ tc_{ij} + p_{ij}\, uc_{ij} + (1 - p_{ij}) C_j^k,\ \beta_i \right\} \right. \\
&\quad \left. - \min_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ tc_{ij} + p_{ij}\, uc_{ij} + (1 - p_{ij}) C_j^{k'},\ \beta_i \right\} \right| \\
&\leq \max_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ \left| (tc_{ij} + p_{ij}\, uc_{ij} + (1 - p_{ij}) C_j^k) \right. \right. \\
&\quad \left. \left. - (tc_{ij} + p_{ij}\, uc_{ij} + (1 - p_{ij}) C_j^{k'}) \right|,\ 0 \right\} \\
&= \max_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ (1 - p_{ij}) \left| C_j^k - C_j^{k'} \right| \right\} \\
&\leq \max_{\forall j, \text{ s.t. } e_{ij} \text{ exists}} \left\{ (1 - p_{min}) \left| C_j^k - C_j^{k'} \right| \right\} \\
&\leq (1 - p_{min}) \max_{1 \leq j \leq n} \left\{ \left| C_j^k - C_j^{k'} \right| \right\} \\
&= \gamma ||\boldsymbol{C^k} - \boldsymbol{C^{k'}}||.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&||\boldsymbol{C^{k+1}} - \boldsymbol{C^{k'+1}}|| \\
&\quad = \max_{1 \leq i \leq n} \left\{ \left| C_i^{k+1} - C_i^{k'+1} \right| \right\} \\
&\quad \leq \gamma ||\boldsymbol{C^k} - \boldsymbol{C^{k'}}||.
\end{aligned}
$$

□

Proof for Lemma 2 Note that because $\forall i, j$ such that e_{ij} exists, $uc_{ij} = uc_{ji} \leq \beta_i$, by Theorem 2, Eq. 1 only presents its first case. By expanding the recursion in Eq. 1, the expected cost of this path is[12]

$$
\begin{aligned}
&C_{\{v_0 \to v_1 \to \cdots\}} \\
&= \sum_{k=1}^{\infty} \left(\left(tc_{k-1,k} + uc_{k-1,k} \cdot p_{k-1,k} \right) \prod_{j=1}^{k-1} \left(1 - p_{j-1,j} \right) \right) + \lim_{k \to \infty} \left(\beta_k \prod_{j=1}^{k} \left(1 - p_{j-1,j} \right) \right) \\
&\leq \sum_{k=1}^{\infty} \left(\left(tc_{max} + uc_{max} \cdot p_{max} \right) \left(1 - p_{min} \right)^{k-1} \right) + \beta_{max} \lim_{k \to \infty} \left(1 - p_{min} \right)^k \\
&= \frac{tc_{max} + uc_{max} \cdot p_{max}}{p_{min}}.
\end{aligned}
$$

□

[12] We let $\prod_{j=1}^{0} \left(1 - p_{j-1,j} \right) = 1$ for the convenience of notation.

Proof for Lemma 4 Using Lemma 1 and the triangle inequality of max norm, we get

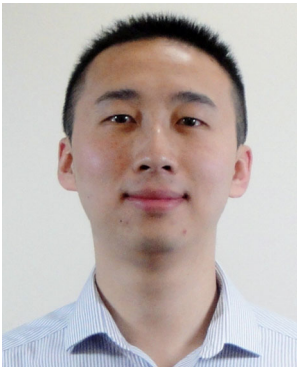
$$\begin{aligned}
&||\boldsymbol{C^{k+1}} - \boldsymbol{C}|| \\
&\quad \leq (1 - p_{min})||\boldsymbol{C^{k}} - \boldsymbol{C}|| \\
&\quad \leq (1 - p_{min})||\boldsymbol{C^{k+1}} - \boldsymbol{C^{k}}|| + (1 - p_{min})||\boldsymbol{C^{k+1}} - \boldsymbol{C}|| \\
&\quad \leq \epsilon_0 \; p_{min} + (1 - p_{min})||\boldsymbol{C^{k+1}} - \boldsymbol{C}||,
\end{aligned}$$

which implies $||\boldsymbol{C^{k+1}} - \boldsymbol{C}|| \leq \epsilon_0$. □

References

1. Ayala D, Wolfson O, Xu B, DasGupta B, Lin J (2011) Parking slot assignment games. In: Proc. of the 19th Intl. conf. on advances in geographic information systems (ACM SIGSPATIAL GIS 2011). Chicago
2. Ayala D, Wolfson O, Xu B, DasGupta B, Lin J (2012) Parking in competitive settings: a gravitational approach. In: Proc. of 13th Intl. conf. on mobile data management (MDM). Bengaluru
3. Ayala D, Wolfson O, Xu B, DasGupta B, Lin J (2012) Spatio-temporal matching algorithms for road networks. In: 20th Int. conf. on advances in geografic information systems (ACM SIGSPATIAL GIS). Redondo Beach
4. Boehlé J, Rothkrantz L, van Wezel M (2008) Cbprs: a city based parking and routing system. ERIM Report Series Reference No. ERS-2008-029-LIS
5. Delot T, Ilarri S, Lecomte S, Cenerario N (2013) Sharing with caution: managing parking spaces in vehicular networks. Mob Inf Syst 9:69–98
6. Geng Y, Cassandras CG (2011) A new "smart parking" system based on optimal resource allocation and reservations. In: Proc. of 14th Intl. conf. on intelligent transportation systems (ITSC). Washington
7. Guo Q, Wolfson O (2015) Presents: probabilistic resource-search networks. In: Proceedings of the 23rd ACM SIGSPATIAL international conference on advances in geographic information systems, SIGSPATIAL '15. ACM
8. Guo Q, Wolfson O (2016) Finding geospatial resources using uncertain data. In: Proc. of 17th Intl. conf. on mobile data management, MDM '16
9. Guo Q, Wolfson O, Ayala D (2015) A framework on spatio-temporal resource search. In: 2015 11th International wireless communications and mobile computing conference (IWCMC)
10. Jossé G, Schmid KA, Schubert M (2015) Probabilistic resource route queries with reappearance. In: Proceedings of the 18th international conference on extending database technology, EDBT '15, pp 445–456
11. Jossé G, Schubert M, Kriegel H-P (2013) Probabilistic parking queries using aging functions. In: Proc. of the 21st ACM SIGSPATIAL Int. conf. on advances in geographic information systems, SIGSPATIAL'13. ACM, Orlando, pp 442–445
12. Kühne R (2003) Potential of remote sensing for traffic applications. In: Proceedings of IEEE intelligent transportation systems
13. Ma S, Wolfson O, Xu B (2014) Updetector: sensing parking/unparking activities using smartphones. In: Proc. of 7th Int. Workshop on computational transportation science (IWCTS)
14. Ma S, Zheng Y, Wolfson O (2013) T-share: a large-scale dynamic taxi ridesharing service. In: Proceedings of the 29th IEEE int. conf. on data engineering (ICDE)
15. Mathur S, Jin T, Kasturirangan N, Chandrashekharan J, Xue W, Gruteser M, Trappe W Parknet: drive-by sensing of road-side parking statistics. In: MobiSys. San Francisco
16. Nawaz S, Efstratiou C, Mascolo C (2013) Parksense: a smartphone based sensing system for on-street parking. In: Proc. of the 19th Annual int. conf. on mobile computing and networking (MobiCom). ACM, pp 75–86
17. Panja B, Schneider B, Meharia P (2011) Wirelessly sensing open parking spaces: accounting and management of parking facility. In: AMCIS 2011 Proceedings
18. Russell S, Norvig P (2003) Artificial intelligence: a modern approach, 2nd edn. Prentice Hall
19. Safra E, Kanza Y, Dolev N, Sagiv Y, Doytsher Y (2007) Computing a k-route over uncertain geographical data. In: Papadias D, Zhang D, Kollios G (eds) Advances in spatial and temporal databases, volume 4605 of lecture notes in computer science. Springer, Berlin Heidelberg, pp 276–293

20. Verroios V, Efstathiou V, Delis A (2011) Reaching available public parking spaces in urban environments using ad-hoc networking. In: IEEE Intl. conf. on mobile data management (MDM)
21. Wolfson O, Xu B (2004) Opportunistic dissemination of spatio-temporal reseource information in mobile peer-to-peer networks. In: Proc. of 1st Int. workshop on P2P data management, security and trust (PDMST'04), DEXA Workshops 2004. Zaragoza, pp 954–958
22. Xu B, Wolfson O, Yang J, Stenneth L, Yu PS (2013) Real time street parking availability estimation. In: Proc. of 14th Intl. conf. on mobile data management (MDM). Milan
23. Yoon JW, Pinelli F, Calabrese F (2012) Cityride: a predictive bike sharing journey advisor. In: Proc. of 13th Intl. conf. on mobile data management (MDM). Bengaluru
24. Yuan NJ, Zheng Y, Zhang L, Xie X (2013) T-finder: a recommender system for finding passengers and vacant taxis. In: IEEE Transactions on knowledge and data engineering



Qing Guo is a PhD candidate in the Department of Computer Science at the University of Illinois at Chicago. His current advisor is Prof. Ouri Wolfson. He received his Bachelor's degree in Statistics at the University of Science and Technology of China in 2008. His research interests include computational transportation science and mobile computing.

Ouri Wolfson, Ph.D., is the Richard and Loan Hill Professor of Computer Science at the University of Illinois at Chicago, and an Affiliate Professor in the Department of Computer Science at the University of Illinois at Urbana Champaign. He is the founder of Mobitrac, a venturefunded high-tech startup that was acquired in 2006, and of Pirouette Software. Wolfson authored over 200 publications, and holds seven patents. He is a Fellow of the ACM, AAAS, IEEE, and a University of Illinois Scholar. He co-authored four award winning papers. Wolfson's main research interests are in databases, distributed systems, mobile/pervasive computing, and intelligent transportation.