Efficient Evaluation of k-NN Queries Using Spatial Mashups[⋆]

Detian Zhang[1,2,3] Chi-Yin Chow[2] Qing Li[2,3] Xinming Zhang[1,3] Yinlong Xu[1,3]

[1]Department of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2]Department of Computer Science, City University of Hong Kong, Hong Kong, China
[3]USTC-CityU Joint Advanced Research Center, Suzhou, China
`tianzdt@mail.ustc.edu.cn {chiychow,itqli}@cityu.edu.hk {xinming,ylxu}@ustc.edu.cn`

Abstract. K-nearest-neighbor (k-NN) queries have been widely studied in time-independent and time-dependent spatial networks. In this paper, we focus on k-NN queries in time-dependent spatial networks where the driving time between two locations may vary significantly at different time of the day. In practice, it is costly for a database server to collect real-time traffic data from vehicles or roadside sensors to compute the best route from a user to an object of interest in terms of the driving time. Thus, we design a new spatial query processing paradigm that uses a spatial mashup to enable the database server to efficiently evaluate k-NN queries based on the route information accessed from an external Web mapping service, e.g., Google Maps, Yahoo! Maps and Microsoft Bing Maps. Due to the expensive cost and limitations of retrieving such external information, we propose a new spatial query processing algorithm that uses shared execution through grouping objects and users based on the road network topology and pruning techniques to reduce the number of external requests to the Web mapping service and provides highly accurate query answers. We implement our algorithm using Google Maps and compare it with the basic algorithm. The results show that our algorithm effectively reduces the number of external requests by 90% on average with high accuracy, i.e., the accuracy of estimated driving time and query answers is over 92% and 87%, respectively.

1 Introduction

With the ubiquity of wireless Internet access, GPS-enabled mobile devices and the advance in spatial database management systems, location-based services (LBS) have been realized to provide valuable information for their users based on their locations [1, 2]. LBS are an abstraction of spatio-temporal queries. Typical examples of spatio-temporal queries include range queries (e.g., "*How*

⋆ The work described in this paper was partially supported by a grant from City University of Hong Kong (Project No. 7200216) and the National Natural Science Foundation of China under Grant 61073185.

many vehicles in a certain area") [3, 4, 5] and k-nearest-neighbor (k-NN) queries (e.g., "*Find the k-nearest gas stations*") [4, 6, 7, 8].

The distance between two point locations in a road network is measured in terms of the network distance, instead of the Euclidean distance, to consider the physical movement constrains of the road network [5]. It is usually defined by the distance of their shortest path. However, this kind of distance measure would hide the fact that the user may take longer time to travel to his/her nearest object of interest (e.g., restaurant and hotel) than other ones due to many realistic factors, e.g., heterogeneous traffic conditions and traffic accidents. Driving time (or travel time) is in reality a more meaningful and reliable distance measure for LBS in road networks [9, 10, 11]. Figure 1 depicts an example where Alice wants to find the nearest clinic for emergency medical treatment. A traditional shortest-path based NN query algorithm returns Clinic X. However, a driving-time based NN query algorithm returns Clinic Y because Alice will spend less time to reach Y (3 mins.) than X (10 mins.).

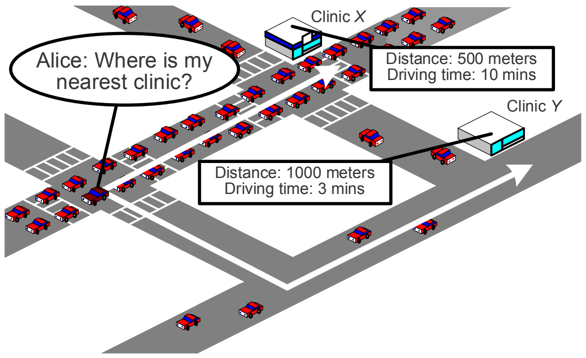


Fig. 1. A shortest-path based NN (X) versus a driving-time based NN (Y).

Since driving time is highly dynamic, e.g., the driving time on a segment of I-10 freeway in Los Angeles, USA between 8:30AM to 9:30AM changes from 30 minutes to 18 minutes, i.e., 40% decrease in driving time [9], it is almost impossible to accurately predict the driving time between two point locations in a road network based on their network distance. The best way to provide real-time driving time computation is to continuously monitor the traffic in road networks; however, it is difficult for every LBS provider to do so due to very expensive deployment cost and privacy issues.

A spatial mashup[1] (or GIS mashup), one of the key technologies in Web 2.0, provides a more cost-effective way to access route information in road networks from external Web mapping services, e.g., Google Maps, Yahoo! Maps, Microsoft Bing Maps and government agencies. However, existing spatial mashups suffer from the following limitations. (1) It is costly to access direction information from a Web mapping service, e.g., retrieving driving time from the Microsoft

[1] A mashup is a web application that combines data, representation, and/or functionality from multiple web applications to create a new application [12].

MapPoint web service to a database engine takes 502 ms while the time needed to read a cold and hot 8 KB buffer page from disk is 27 ms and 0.0047 ms, respectively [13]. (2) There is usually a limit on the number of requests to a Web mapping service, e.g., Google Maps allows only 2,500 requests per day for evaluation users and 100,000 requests per day for premier users [14]. (3) The use of retrieved route information is restricted, e.g., the route information must not be pre-fetched, cached, or stored, except only limited amount of content can be temporarily stored for the purpose of improving system performance [14]. (4) Existing Web mapping services only support primitive operations, e.g., the driving direction and time between two point locations. A database server has to issue a large number of external requests to collect small pieces of information from the supported simple operations to process relatively complex spatial queries, e.g., k-NN queries.

In this paper, we design an algorithm to processing k-NN queries using spatial mashups. Given a set of objects and a k-NN query with a user's location and a user specified maximum driving time t_{max} (e.g., "*Find the k-nearest restaurants that can be reached in 10 minutes by driving*"), our algorithm finds at most k objects with the shortest driving time and their driving time is no longer than t_{max}. The objectives of our algorithm are to reduce the number of external requests to a Web mapping service and provide query answers with high accuracy. To achieve our objectives, we use shared execution by grouping objects based on the road network topology and pruning techniques to reduce the number of external requests. We design two methods to group objects to adjust a performance trade-off between the number of external requests and the accuracy of query answers. We first present our algorithm in road networks with bidirectional road segments, and then adapt the algorithm to road networks with both one- and two-way road segments. In addition, we design another extension to further reduce the number of external requests by grouping users based on their movement direction and the road network topology for a system with high workloads or a large number of continuous k-NN queries.

To evaluate the performance of our algorithm, we build a simulator to compare it with a basic algorithm in a real road network. The results show that our algorithms outperform the basic algorithm in terms of the number of external requests and query response time. We also implement our algorithm using Google Maps [15]. The experimental results show that our algorithm provides highly accurate k-NN query answers.

The remainder of this paper is organized as follows. Section 2 describes the system model. Section 3 presents the basic algorithm and our algorithm. Section 4 gives two extensions to our algorithm. Simulation and experimental results are analyzed in Section 5. Section 6 highlights related work. Finally, Section 7 concludes this paper.

2 System Model

In this section, we describe our system architecture, road network model, and problem definition. Figure 2 depicts our system architecture that consists of three entities, users, a database server, and a Web mapping service provider. Users send k-NN queries to the database server at a LBS provider at anywhere and anytime. The database server processes queries based on local data (e.g., the location and basic information of restaurants) and external data (i.e., routes and driving time) accessed from the Web mapping service. In general, accessing external data is much more expensive than accessing internal data [13].

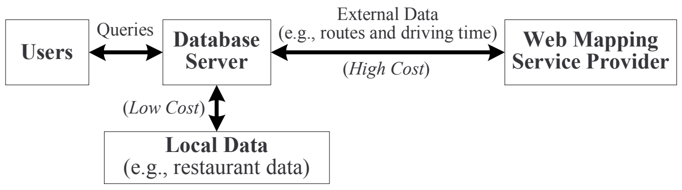
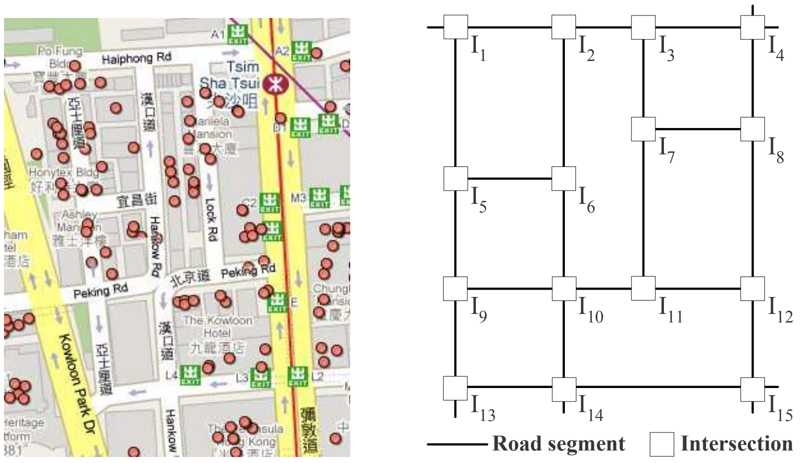

Users
Queries
Database Server
External Data (e.g., routes and driving time)
(High Cost)
Web Mapping Service Provider
(Low Cost)
Local Data (e.g., restaurant data)

Fig. 2. System architecture.

We use a graph $G = (V, E)$ to model a road network, where E and V are a set of road segments and intersections of road segments, respectively. For example, Figure 3a depicts a real road map that is modeled into an undirected graph (Figure 3b), where an edge represents a road segment (e.g., I_1I_2 and I_1I_5) and a square represents an intersection (e.g., I_1 and I_2). This road network model will be used in Section 3 because we assume that each road segment is bidirectional. In Section 4, we will use a directed graph to model a road network where each edge with an *arrow* or *double arrows* to indicate that the corresponding road segment is *one-way* or *two-way*, respectively (e.g., Figure 9).

Our problem is defined as follows. Given a set of objects O and a NN query $Q = (\lambda, k, t_{max})$ from a user U, where λ is U's location, k is U's specified maximum number of returned objects, and t_{max} is U's required maximum driving time from λ to a returned object, our system returns U at most k objects in O



(a) A road map (b) A graph model

Fig. 3. Road network model.

with the shortest driving time from λ and their driving time must be no longer than t_{max}, based on the routes and driving time accessed from a Web mapping service, e.g., Goolge Maps [15]. Since accessing the Web mapping service is expensive, our objectives are to reduce the number of external requests to the Web mapping service and provide highly accurate query answers.

3 Processing *k*-NN Queries Using Spatial Mashups

In this section, Section 3.1 describes a basic algorithm to process k-NN queries using spatial mashups. Then, Section 3.2 presents our efficient algorithm that aims to minimize the number of external requests to a Web mapping service by using shared execution and pruning techniques.

3.1 Basic Algorithm

Since there could be a very large number of objects in a spatial data set, it is extremely inefficient to issue an expensive external request to the Web mapping service to retrieve the route information and driving time from a user to each object in the data set. To reduce the number of external requests, the basic algorithm executes a range query in spatial networks [5] to prune the whole data set into a much smaller set of *candidate objects* that are within the maximum possible driving distance, $dist_{max}$, from the user. The most conservative way to compute $dist_{max}$ is to multiply the user-specified maximum driving time t_{max} by the maximum allowed driving speed of the road network. Since only the candidate objects can be reached by the user within the driving time of t_{max}, the database server only needs to issue one external request to the Web mapping service for each candidate object to access the route and driving time from the user to the object.

Figure 4 shows an example for the basic algorithm, where a user U is represented by a triangle, 10 objects R_1 to R_{10} are represented by circles, and the maximum allowed driving speed is 80 km/h. If U wants to find the nearest object within a driving time of five minutes (i.e., $k = 1$ and $t_{max} = 5$ mins.), $dist_{max}$ is 6.7 km. The database server executes a range query with a range distance of 6.7 km, where the road segments within the range distance are highlighted. Thus, the whole data set is pruned into a subset of seven candidate objects R_1, R_2, R_4, R_5, R_8, R_9 and R_{10}; the database server only needs to issue seven external requests rather than ten requests.

3.2 Our Efficient *k*-NN Query Processing Algorithm

Although the basic algorithm can reduce the whole data set into a much smaller subset, if the user-specified maximum driving time is very long or the object density in the vicinity of the user is very high, the database server may still need to issue a large number of external requests to the Web mapping service. To this end, we propose a new k-NN query processing algorithm that utilizes shared execution and pruning techniques to further reduce the number of external requests.

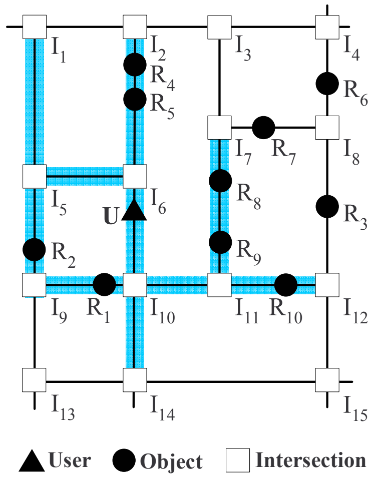


Fig. 4. Basic algorithm.

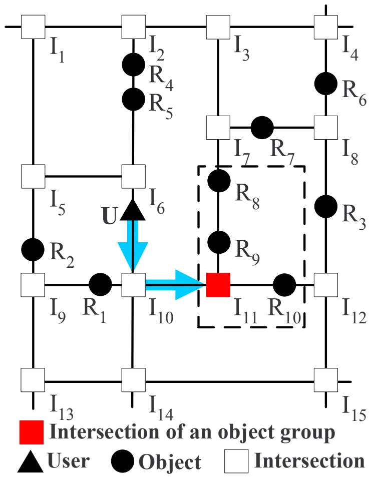


Fig. 5. Object grouping.

Overview. Our algorithm has four main steps. (1) Our algorithm takes a set of candidate objects computed by the basic algorithm as an input (Section 3.1). (2) It selects representative points in the road network and clusters objects to them to form groups (Section 3.2.1). (3) It issues an external request for each group to retrieve the route information from the user to the corresponding representative point and estimates the driving time from the user to each object in the group (Section 3.2.2). (4) The algorithm prunes candidate objects that cannot be part of a query answer (Section 3.2.3). The control flow of our algorithm is as follows. After it performs steps (1) and (2), it repeats steps (3) and (4) until all the candidate objects are processed or pruned.

3.2.1 Grouping Objects for Shared Execution

We observe that many spatial objects are generally located in clusters in real world. For example, many restaurants are located in a downtown area (Figure 3a). Thus, it makes sense to group nearby objects to a representative point. The database server issues only one external request to the Web mapping service for an object group. Then, it shares the retrieved route and driving time information from a user to the representative point among the objects in the group to estimate the driving time from the user to each object. This object grouping technique reduces the number of external requests from the number of candidate objects in the basic algorithm to the number of groups (at the worst case) for a k-NN query. Section 3.2.3 will describe another optimization to further reduce the number of external requests.

There are two challenges in grouping objects: (a) How to select representative points in a road network? and (b) How to group objects to representative points? Our solution is to select intersections as representative points in a road network and group objects to them. The reason is twofold: (1) Since there is only one possible path from the intersection of a group $\mathcal{G}$ to each object in $\mathcal{G}$, it is easy to estimate the driving time from the intersection to each object in $\mathcal{G}$. (2) A road segment should have the same conditions, e.g., speed limit and direction. The estimation of driving time would be more nature and accurate by considering a road segment as a basic unit [16].

Figure 5 gives an example for object grouping where objects R_8, R_9 and R_{10} are grouped together by intersection I_{11}. The database server only needs one external query from user U to I_{11} to the Web mapping service to retrieve the route information and driving time from U and I_{11}. Then, our algorithm estimates the driving time from I_{11} to each of its group members.

In the reminder of this section, we will present two methods for grouping objects. These methods have different performance trade-offs between the number of external requests and the accuracy of query answers. Section 3.2.2 (the third step) will discuss how to perform driving time estimation and Section 3.2.3 (the forth step) will discuss how to prune intersections safely to further reduce the number of external requests.

Method 1: Minimal intersection set (MinIn). To minimize the number of external requests to the Web mapping service, we should find a minimal set of intersections that cover all road segments having candidate objects, which are found by the basic algorithm. Given a k-NN query issued by a user U, we convert the graph G of the road network model into a subgraph $G_u = (V_u, E_u)$ such that E_u is an edge set where each edge contains at least one candidate object and V_u is a vertex set that consists of the vertices (intersections) of the edges in E_u. The result vertex set $V'_u \subseteq V_u$ is a minimal one covering all the edges in E_u. This problem is the vertex cover problem [17], which is NP-complete, so we use a greedy algorithm to find V'_u.

In the greedy algorithm, we calculate the number of edges connected to a vertex v as the degree of v. The algorithm selects the vertex from V_u with the largest degree to V'_u. In case of a tie, the vertex with the shortest distance to U is selected to V'_u. The selected vertex is removed from V_u and the edges of the selected vertex are removed from the E_u. After that, the degree of the vertices of the removed edges is updated accordingly.

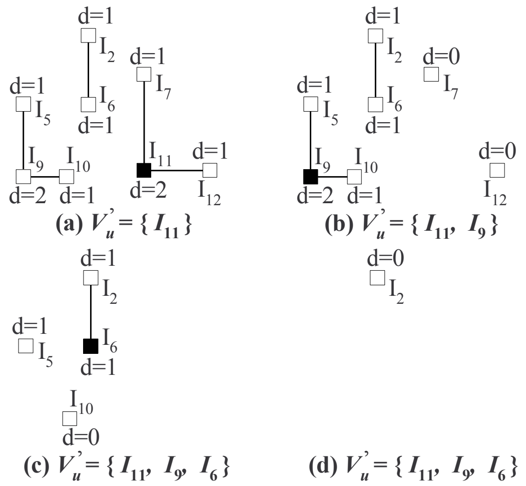
Figure 6 gives an example for the MinIn method. After the basic algorithm finds a set of candidate objects (represented by black circles in Figure 7). The MinIn method first constructs a subgraph $G_u = (V_u, E_u)$ from G. Since edges I_2I_6, I_5I_9, I_9I_{10}, I_7I_{11} and $I_{11}I_{12}$ contain some candidate objects, these five edges constitute E_u and their vertices constitute V_u. Then, the MinIn method calculates the degree for each vertex in V_u. For example, the degree of I_{11} is two because two edges I_7I_{11} and $I_{11}I_{12}$ are connected to I_{11}. Figure 6a shows that I_{11} has the largest degree and is closer to U than I_9, I_{11} is selected to V'_u and the edges connected to I_{11} are removed from G_u. Figure 6b shows that the degree of the vertices I_7 and I_{12} of the two deleted edges is updated accordingly. Any vertex in G_u without any connected edge is removed from V_u immediately. Since I_9 has the largest degree, I_9 is selected to V'_u and the edges connected to I_9 are removed. Similarly, I_6 is selected to V'_u (Figure 6c). Since I_2 has no connected edge, I_2 is removed from V_u. After that, V_u becomes empty, so the MinIn method clusters the candidate objects into three groups (indicated by dotted rectangles), $I_6 = \{R_4, R_5\}$, $I_9 = \{R_1, R_2\}$, and $I_{11} = \{R_8, R_9, R_{10}\}$ (Figure 7).

Method 2: Nearest intersections (NearestIn). Although the MinIn method can minimize the number of external requests to the Web mapping service, it may lead to low accuracy in the estimated driving time for some objects. For example, consider object R_1 in Figure 7, since R_1 is grouped to intersection I_9 and I_{10} is not selected to V'_u, the MinIn method will result in a path $P_1 : U \rightarrow I_6 \rightarrow I_5 \rightarrow I_9 \rightarrow R_1$. However, another path $P_2 : U \rightarrow I_{10} \rightarrow R_1$ may give shorter driving time than P_1. To this end, we design an alternative method, NearestIn, that groups objects to their nearest intersections. Figure 8 gives an example for NearestIn, where the candidate objects are represented by black circles. Since R_2 is closer to intersection I_9 than I_5, R_2 is grouped to I_9. The NearestIn method constructs six groups (indicated by dotted rectangles), $I_9 = \{R_2\}$, $I_{10} = \{R_1\}$, $I_2 = \{R_4, R_5\}$, $I_7 = \{R_8\}$, $I_{11} = \{R_9\}$, and $I_{12} = \{R_{10}\}$; thus, the database server needs to issue six external queries.

In general, NearestIn gives more accurate driving time than MinIn, but NearestIn needs more external requests than MinIn. As in our running examples, MinIn and NearestIn need to issue three and six external requests (one request per group), respectively. Thus, these methods provide different performance tradeoffs between the overhead of external requests and the accuracy of driving time estimation. We verify their performance tradeoffs in Section 5.

3.2.2 Calculating Driving Time for Grouped Objects

Most Web mapping services, e.g., Goolge Maps, return turn-by-turn route information for a request. For example, if the database server sends a request to the Web mapping service to retrieve the route information from user U to intersection I_9 in Figure 7, the service returns the turn-by-turn route information, i.e., $Route(U \rightarrow I_9) = \{\langle d(U \rightarrow I_6), t(U \rightarrow I_6)\rangle, \langle d(I_6 \rightarrow I_5), t(I_6 \rightarrow I_5)\rangle,$



(a) $V'_u = \{I_{11}\}$
(b) $V'_u = \{I_{11}, I_9\}$
(c) $V'_u = \{I_{11}, I_9, I_6\}$
(d) $V'_u = \{I_{11}, I_9, I_6\}$
d=1
d=2
d=0

Fig. 6. Greedy algorithm for MinIn.

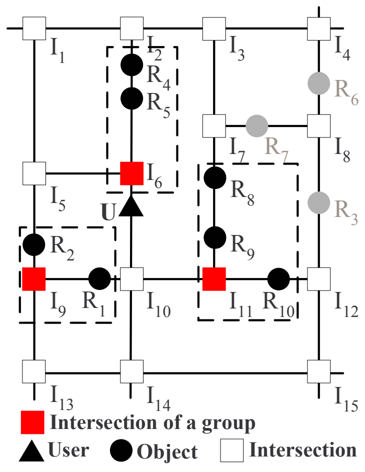


Fig. 7. MinIn object grouping.

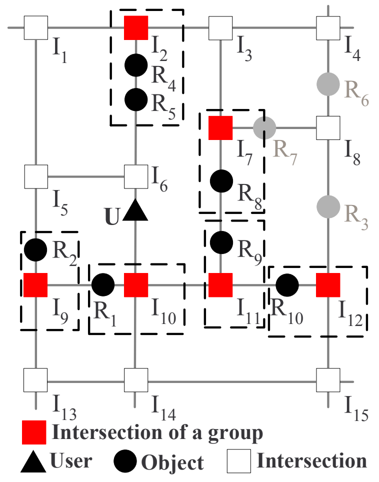


Fig. 8. NearestIn object grouping.

$\langle d(I_5 \rightarrow I_9), t(I_5 \rightarrow I_9)\rangle\}$, where $d(A \rightarrow B)$ and $t(A \rightarrow B)$ are the Euclidean distance and driving time from location point A to location point B, respectively.

After grouping objects to intersections, each intersection in V'_u is processed based on its distance to the user in ascending order. The reason is that knowing the driving time of more objects closer to the user at an earlier stage gives more pruning power to the next step (Section 3.2.3). If a group contains only one object, the database server simply issues an external request to retrieve the route information from the user to the object. Otherwise, our algorithm calculates the driving time for each object and selects the best route (if appropriate) as follows.

Driving time calculation. Based on whether a candidate object is on the last road segment of a retrieved route, we can distinguish two cases.

Case 1: An object is on the last road segment. In this case, we assume that the speed of the last road segment is constant. For example, given the information of the route from U to I_9 (i.e., $Route(U \rightarrow I_9)$) retrieved from the Web mapping service, object R_2 is on the last road segment of the route, i.e., $I_5 I_9$. Hence, the driving time from U to R_2 is calculated as: $t(U \rightarrow R_2) = t(U \rightarrow I_9) - t(I_5 \rightarrow I_9) \times \frac{d(R_2 \rightarrow I_9)}{d(I_5 \rightarrow I_9)}$.

Case 2: An object is NOT on the last road segment. In this case, a candidate object is not on a retrieved route, i.e., the object is on a road segment S connected to the last road segment S' of the route. We assume that the driving speed of S is the same as that of S'. For example, given the retrieved information of the route from U to I_9 (i.e., $Route(U, I_9)$), object R_1 is not on the last road segment of the route, i.e., $I_5 I_9$. Hence, the driving time from U to R_1 is calculated as: $t(U \rightarrow R_1) = t(U \rightarrow I_9) + t(I_5 \rightarrow I_9) \times \frac{d(I_9 \rightarrow R_1)}{d(I_5 \rightarrow I_9)}$.

Route selection. The object grouping methods may select both the intersections of a road segment for a candidate object. In this case, the database server finds the route from the querying user to the object via each of the intersections. Thus, our algorithm selects the route with the shortest driving time. For example, since both the intersections of edge I_9I_{10}, i.e., I_9 and I_{10}, are selected (Figure 8), the driving time from user U to R_1 is selected as $t(U \to R_1) = \min(d(U \to I_9 \to R_1), d(U \to I_{10} \to R_1))$.

3.2.3 Object Pruning

After the algorithm finds a current answer set A (i.e., the best answer so far) for a user U's k-NN query, this step keeps track of the longest driving time A_{max} from U to the objects in A, i.e., $A_{max} = \max\{t(U \to R_i) | R_i \in A\}$. It uses A_{max} to prune the candidate objects to further reduce the number of external requests to the Web mapping service. The basic idea is that a candidate object can be pruned safely if the smallest possible driving time from U to the object is not shorter than A_{max}, because the object cannot be part of a query answer. The smallest possible driving time of a candidate object is calculated by dividing the distance of the shortest path from the user to the object by the maximum allowed driving speed of the underlying road network. Whenever A_{max} is updated, this step checks all unprocessed candidate objects and prunes objects that cannot be part of the query answer. After removing an object from a group, if the group becomes empty, the corresponding intersection is also removed from the vertex set V'_u.

4 Extensions

In this section, we present two extensions to our advanced k-NN query processing algorithm. The first extension enables our algorithm to support one-way road segments (Section 4.1). Our algorithm with the second extension can group users for shared execution to further reduce the number of external requests (Section 4.2).

4.1 One-Way Road Segments

In real world, a street could be only one-way, so it is essential to extend our algorithm to support one-way road segments. The basic algorithm can be easily

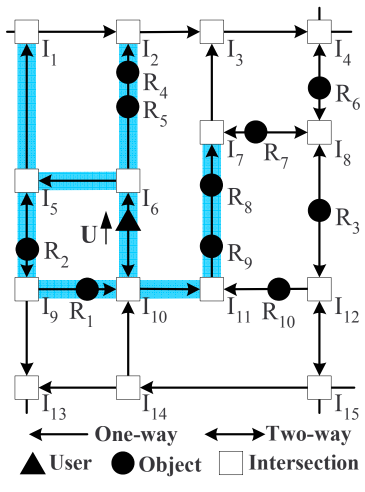


Fig. 9. Basic algorithm.

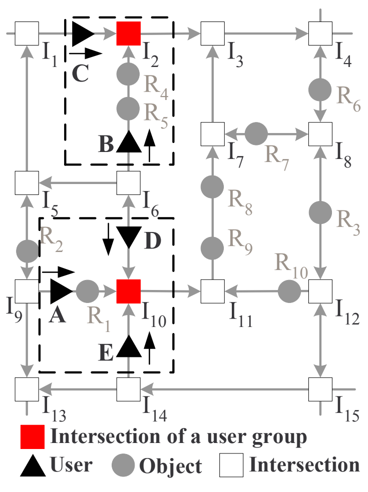


Fig. 10. User grouping.

extended to support one-way streets by using a directed graph to model a road network. Figure 9 depicts an example where the objects on the highlighted road segments are probably reached from user U with the user-specified maximum driving time t_{max}. In this example, edges $I_{10}I_{14}$ and $I_{11}I_{12}$ are no longer considered, compared to the road network with only bi-directional road segments depicted in Figure 4.

To enable our algorithm to support one-way road segments, we only need to slightly modify the object grouping methods.

The MinIn method. After the basic algorithm finds a set of candidate objects, the MinIn method constructs a directed graph $G_u = (V_u, E_u)$ where E_u is a set of road segments containing some candidate objects and V_u is set of vertices of the edges in E_u. The degree of each vertex in V_u is calculated by the number of its outgoing edges in E_u. The basic idea of the greedy algorithm is that a vertex in V_u with the highest degree is selected to the result vertex set V_u'. Then, the edges of the selected vertex are removed from E_u and the degree of other adjacent vertices are updated accordingly. Any vertex with no edges is also removed from V_u.

Figure 12 depicts an example, where six candidate objects are found by the basic algorithm. Since there are four edges containing some candidate objects, we construct a directed graph G_u with $E_u = \{I_5I_9, I_9I_{10}, I_2I_6, I_7I_{11}\}$ and $V_u = \{I_2, I_5, I_6, I_7, I_9, I_{10}, I_{11}\}$. Since I_9 has two outgoing edges I_9I_5 and I_9I_{10}, the degree of I_9 is two (Figure 11a). Figure 11a shows that I_9 has the largest degree, I_9 is selected to V_u' and I_9 is removed from V_u. After the edges of I_9 are removed, I_5 and I_{10} have no more edges, so they both are deleted from V_u. Then, since I_6 is closer to U than I_{11}, I_6 is selected (Figure 11b). I_{11} is next to be selected to V_u' (Figure 11c). After deleting I_7, V_u becomes empty (Figure 11d), so the MinIn method is done. The candidate objects are grouped into three groups indicated by dotted rectangles, i.e., $I_9 = \{R_1, R_2\}$, $I_6 = \{R_4, R_5\}$, and $I_{11} = \{R_8, R_9\}$, as illustrated in Figure 12.

The NearestIn method. Since a candidate object on a one-way road segment can only be reached from a user through its starting intersection, the object is simply grouped to the starting intersection. For example, object R_1 is grouped to I_9, in Figure 12. However, a candidate object is grouped to the nearest intersection for a two-way road segment.

4.2 Grouping Users for Shared Execution

To further improve the system performance of a database server with a very high workload, e.g., a large number of users or continuous queries, we design another shared execution for grouping users. Similar to object grouping, we consider intersections in the road network as representative points. The database server only needs to issue one external request from the intersection of a user group $\mathcal{G}_u$ to the intersection of an object group $\mathcal{G}_o$ to estimate the driving time from each user U_i in $\mathcal{G}_u$ to each object R_j in $\mathcal{G}_o$. In general, our algorithm with the user grouping extension has five main steps.

User grouping. The key difference between user grouping and object grouping is that users are moving. Grouping users has to consider their movement

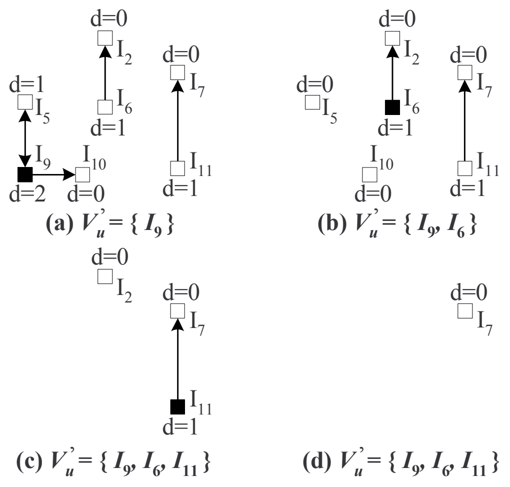

(a) $V_u' = \{I_9\}$
(b) $V_u' = \{I_9, I_6\}$
(c) $V_u' = \{I_9, I_6, I_{11}\}$
(d) $V_u' = \{I_9, I_6, I_{11}\}$
d=0
d=1
d=2

Fig. 11. Greedy algorithm for MinIn.

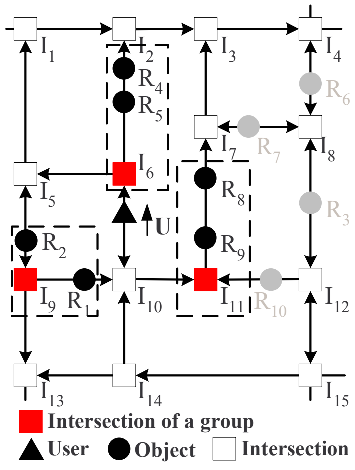


Fig. 12. MinIn object grouping.

direction, so a user is grouped to the nearest intersection to which the user is moving. Figure 10 depicts an example, where user A on edge I_9I_{10} is moving towards I_{10}, so A is grouped to I_{10}. Similarly, users D and E are also grouped to I_{10}. Users C and B on edges I_1I_2 and I_6I_2, respectively, are both moving to I_2, so they are grouped to I_2.

Candidate objects. This step uses the basic algorithm (described in Section 3.1) to find a set of candidate objects $\mathcal{R}$ for a user group $\mathcal{G}_u$. Since the users in $\mathcal{G}_u$ may have different user-required maximum driving times, the basic algorithm finds the candidate objects that are within the largest required maximum driving time of the intersection of $\mathcal{G}_u$. Consider the user group of intersection I_{10} in Figure 10, if the required maximum driving times of users A, D, and E are 5, 10, and 20 minutes, respectively, the basic algorithm finds a set of candidate objects within 20 minutes driving time from I_{10}.

Object grouping. This step employs one of the object grouping methods presented in Section 3.2.1 to group candidate objects in $\mathcal{R}$ to intersections.

Driving time calculation. For each user group $\mathcal{G}_u$, our algorithm processes the object groups one by one based on the distance between their intersection and the intersection of $\mathcal{G}_u$ in ascending order. Such a processing order not only gives more pruning power to the object pruning step, but it also provides a fair response time for the users in $\mathcal{G}_u$. When our algorithm finds that a candidate object cannot be part of answers of any users in $\mathcal{G}_u$, the object is pruned. Similarly, when we guarantee that the remaining candidate objects in $\mathcal{R}$ cannot be part of a user's answer, the user's current answer is returned to the user without waiting the completion of processing all the queries issued by the users in $\mathcal{G}_u$. After the database server retrieves the route and driving time information from the intersection of $\mathcal{G}_u$ (I_u) to the intersection of an object group $\mathcal{G}_o$ (I_o), this step computes the driving time from I_u to each object R_j in $\mathcal{G}_o$, i.e., $t(I_u \rightarrow R_j)$, as presented in Section 3.2.2. Then, this step estimates the driving time from each user U_i in $\mathcal{G}_u$ to I_u, i.e., $t(U_i \rightarrow I_u)$. Since a user is grouped to an intersection to which the user is moving, the driving time from the user U_i to I_u has to be added to the driving time from I_u to each object R_j in $\mathcal{G}_o$. The user is either on the first road segment I_uI_p of the route retrieved from the Web mapping service or on another road segment I_uI_q connected to I_uI_p, so we use the driving speed of I_uI_p to estimate the required diving time by $t(U_i \rightarrow I_u) = t(I_u \rightarrow I_p) \times \frac{d(U_i \rightarrow I_u)}{d(I_u \rightarrow I_p)}$; hence, the driving time from U_i to R_j is $t(U_i \rightarrow R_j) = t(U_i \rightarrow I_u) + t(I_u \rightarrow R_j)$.

Object pruning. After the algorithm finds a current answer set A_i (i.e., the best answer so far) for each user U_i in a user group $\mathcal{G}_u$, this step keeps track of the longest driving time A_{max_i} from U_i to the objects in A_i and the smallest possible driving time $A_{min_{\mathcal{G}}}(R_j)$ from any user U_i in $\mathcal{G}_u$ to an unprocessed candidate object $R_j \in \mathcal{R}$. $A_{min_{\mathcal{G}}}(R_j)$ is calculated by dividing the distance of the shortest path from U_i to R_j by the maximum allowed driving speed of the underlying road network. The step finds the largest value of A_{max_i} of $\mathcal{G}_u$, i.e., $A_{max_{\mathcal{G}}} = \max\{A_{max_i} | U_i \in \mathcal{G}_u\}$. For a candidate object R_j in $\mathcal{R}$, if $A_{min_{\mathcal{G}}}(R_j) \geq A_{max_{\mathcal{G}}}$, R_j is pruned from $\mathcal{G}_u$ because R_j cannot be part of any query answer. Whenever $A_{max_{\mathcal{G}}}$ is updated, this step checks the candidate objects in $\mathcal{R}$. If an object group becomes empty, the intersection of the object group is removed from V'_u. If $A_{max_i} < \min\{A_{min_{\mathcal{G}}}(R_j) | R_j \in \mathcal{R}\}$, any unprocessed candidate object cannot be part of U_i's query answer; thus, U_i's current answer is returned to U_i and U_i is removed from $\mathcal{G}_u$. The processing of $\mathcal{G}_u$ is done if all the intersections in V'_u have been processed/pruned or $\mathcal{G}_u$ becomes empty.

4.3 Performance Analysis

We now evaluate the performance of our algorithm. Let M and N be the number of users and the number of objects in the database server, respectively. Without any optimization, a naive algorithm issues N external requests for each user; hence, the total number of external requests is $Cost_N = N \times M$. By using the basic algorithm, the database server executes a range query to find a set of candidate objects for each user. Suppose the number of candidate objects for user U_i is α_i; the total number of external requests of M users is $Cost_B = \sum_{i=1}^{M} \alpha_i$. Since usually $\alpha_i \ll N$, $Cost_B \ll Cost_N$.

Our efficient algorithm further uses object- and user-grouping shared execution schemes to reduce the number of external requests. For each user group $\mathcal{G}_j$, our algorithm uses one of the object grouping methods to group its candidate objects. Suppose that the number of user groups is m and the number of object groups is β_j. The total number of required external requests is $Cost_E = \sum_{j=1}^{m} \beta_j$. In general, $m \ll M$ and $\beta_i \ll \alpha_i \ll N$, so $Cost_E \ll Cost_B \ll Cost_N$. We will confirm our performance analysis through the experiments in Section 5.

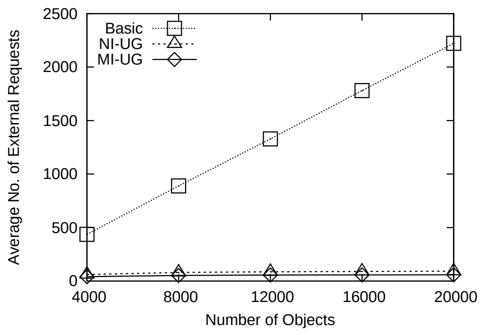
5 Performance Evaluation

In this section, we evaluate our efficient k-NN query processing algorithm using spatial mashups in a real road network of Hennepin County, MN, USA. We select a square area of 8×8 km^2 that contains 6,109 road segments and 3,593 intersections, and the latitude and longitude of its left-bottom and right-top corners are (44.898441, -93.302791) and (44.970094, -93.204015), respectively. The maximum allowed driving speed is 110 km per hour. In all the experiments, we compare our advanced algorithm with user grouping (UG) and object grouping, including MinIn and NearestIn, which are denoted as MI-UG and NI-UG, respectively, with the basic algorithm. We first evaluate the performance of our algorithm through a large-scale simulation (Section 5.1), and then evaluate its accuracy through an experiment using Google Maps [15] (Section 5.2).

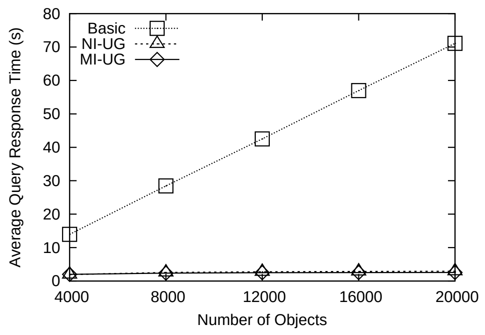
5.1 Simulation Results

Unless mentioned otherwise, we generate 10,000 objects and 10,000 users that are uniformly distributed in the road network for all the simulation experiments. The default user required maximum driving time (t_{max}) is 120 seconds and the requested number of nearest objects (k) is 20. We measure the performance of our algorithm in terms of the average number of external requests per user to the Web mapping service and the average query response time per user.

Effect of the number of objects. Figure 13 depicts the performance of our algorithms with respect to increasing the number of objects from 4,000 to 20,000. Our algorithms, i.e., MI-UG and NI-UG, outperform the basic algorithm. The performance of our algorithms is only slightly affected by the increase of the number of objects (Figure 13a). The results confirm that our algorithms can scale up to a large number of objects. Figure 13b shows the average query response time of our algorithms. The average query response time is the sum of the average query processing time of the algorithm and the multiplication of the average number of external queries per user and the average response time per external request. The average response time per external request is 32 milliseconds that is derived from the experiments (Section 5.2).



(a) Number of external requests



(b) Query response time

Fig. 13. Number of objects.

Effect of the number of users. Figure 14 gives the performance of our algorithms with an increase of the number of users from 4,000 to 20,000. The results also show that our algorithms outperform the basic algorithm. It is expected that MI-UG needs smaller numbers of external requests than NI-UG. Since our algorithms effectively group users to intersections for shared execution, when there are more users, the number of external requests reduces.

Effect of the user-required maximum driving time. Figure 15 shows the performance of our algorithms with various user-required maximum driving times (t_{max}) that are increased from a range of [60, 120] to [60, 600] seconds. When t_{max} gets longer, more candidate objects can be reached by the user; thus, the number of external requests increases. The results also indicate that the increase rate of our algorithms is much smaller than that of the basic algorithm, so our algorithms significantly improve the system scalability.

In summary, all the simulation results consistently show that our algorithms outperform the basic algorithms, in terms of both the number of external requests to the Web mapping service and the query response time. The results also confirm that our algorithms effectively scale up to a large number of objects, a large number of users, and long user-required maximum driving times.

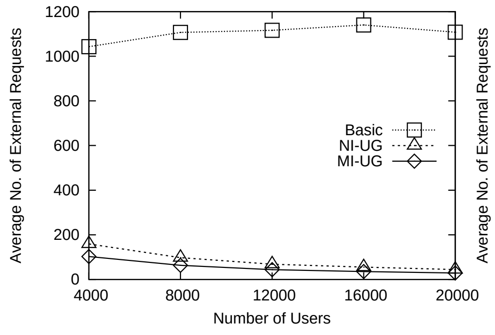


Fig. 14. Number of users.

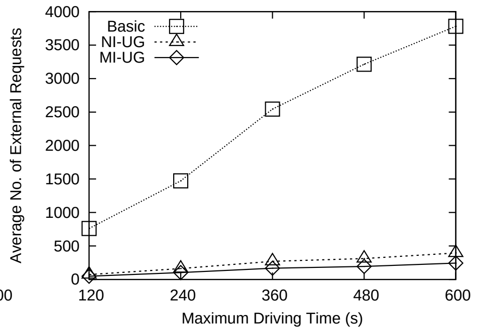


Fig. 15. Required maximum driving time.

5.2 Experiment Results

To evaluate the accuracy of our query processing algorithm, we implement it with the proposed object and user grouping schemes using the Google Maps [15]. Because Google Maps allows only 2,500 requests per day for evaluation users, all the experiments in this section contain 100 users and 500 objects that are uniformly distributed in the underlying road network. We evaluate the accuracy of the driving time estimation and the accuracy of k-NN query answers.

Accuracy of the driving time estimation. The first experiment evaluates the accuracy of the driving time estimation used in object- and user-grouping methods with respect to varying the user required maximum driving time t_{max} from 120 to 600 seconds, as depicted in Figure 16. We compare the accuracy of the driving time estimation of our algorithms MI-UG and NI-UG with the basic algorithm which retrieves the driving time from each user to each object from the Google Maps directly and finds query answers. The accuracy of an estimated driving time is computed by:

$$Accuracy\ of\ estimated\ driving\ time\ (Acc_{time}) = 1 - \min\left(\frac{|\widehat{T} - T|}{T}, 1\right), \quad (1)$$

where T and $\widehat{T}$ are the actual driving time (retrieved by the basic algorithm) and the estimated one, respectively, and $0 \leq Acc_{time} \leq 1$. The results show that the accuracy of our algorithms is at least 0.92; our algorithms achieve highly accurate driving time estimation. Since MI-UG generates the smallest number of external requests, its accuracy is worse than NI-UG.

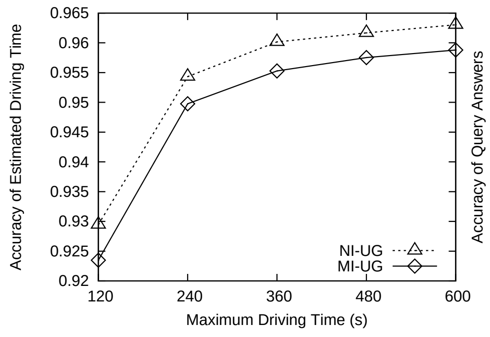


Fig. 16. Accuracy of driving time.

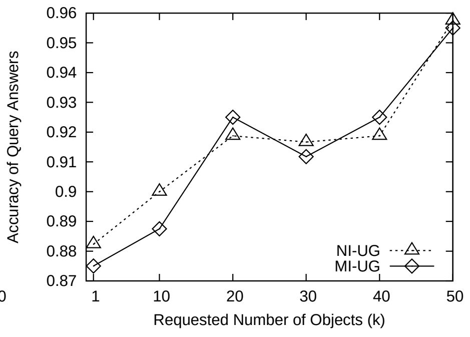


Fig. 17. Accuracy of query answers.

Accuracy of query answers. The second experiment evaluates the accuracy of k-NN query answers returned by our algorithms MI-UG and NI-UG with respect to increasing the required number of nearest objects (k) from 1 to 50. The accuracy of a query answer returned by our algorithms is calculated by:

$$Accuracy\ of\ a\ query\ answer\ (Acc_{ans}) = \frac{|\widehat{A} \cap A|}{|A|}, \quad (2)$$

where A is an exact query answer returned by the basic algorithm and $\widehat{A}$ is a query answer returned by our algorithms and $0 \leq Acc_{ans} \leq 1$. Figure 17 shows that our algorithms can provide highly accurate query answers. When $k = 1$, the accuracy of all our algorithms is over 87%. When k increases, the accuracy of our algorithms improves. Therefore, the results confirm that our algorithms not only effectively reduce the number of external requests to the Web mapping service, but they also provide highly accurate k-NN query answers.

6 Related Work

Existing location-based query processing algorithms can be categorized into two main classes: location-based queries in *time-independent spatial networks* and in *time-dependent spatial networks*. (1) *Time-independent spatial networks.* In this class, location-based query processing algorithms assume that the cost or weight of a road segment, which can be in terms of distance or travel time, is constant (e.g., [5, 18, 19]). These algorithms mainly rely on pre-computed distance or travel time information of road segments in road networks. However, the actual travel time of a road segment may vary significantly during different times of the day due to dynamic traffic on road segments [9, 10]. (2) *Time-dependent spatial networks.* The location-based query processing algorithms designed for time-dependent spatial networks have the ability to support dynamic weights of road segments and topology of a road network, which can change with time. Location-based queries in time-dependent spatial networks are more realistic but also more challenging. George et al. [11] proposed a time-aggregated graph, which uses time series to represent time-varying attributes. The time-aggregated graph can be used to compute the shortest path for a given start time or to find the best start time for a path that leads to the shortest travel time. In [9, 10], Demiryurek et al. proposed solutions for processing k-NN queries in time-dependent road networks where the weight of each road segment is a function of time.

In this paper, we focus on k-nearest-neighbor (NN) queries in time-dependent spatial networks. Our work distinguishes from previous work [9, 10, 11] is that our focus is not on modeling the underlying road network based on different criteria. Instead, we rely on third party Web mapping services, e.g., Google Maps, to compute the travel time of road networks and provide the direction and driving time information through spatial mashups. Since the use of external requests to access route information from a third party is more expensive than accessing local data [13], we propose a new k-NN query processing algorithm for spatial mashups that uses shared execution and pruning techniques to reduce the number of external requests and provide highly accurate query answers.

There are some query processing algorithms designed to deal with expensive attributes that are accessed from external Web services (e.g., [13, 20, 21]). To minimize the number of external requests, these algorithms mainly focused on using either some *cheap* attributes that can be retrieved from local data sources [13, 20] or sampling methods [21] to prune candidate objects, and they only issue absolutely necessary external requests. The closest work to this paper is [13], where Levandoski et al. developed a framework for processing spatial skyline queries by pruning candidate objects that are guaranteed not to be part of a query answer and only issuing an external request for each remaining candidate object to retrieve the driving time from the user to the object. However, all previous work did not study how to use shared execution techniques and the road network topology to reduce the number of external requests to external Web services that is exactly the problem that we solved in this paper.

7 Conclusion

In this paper, we proposed a new k-nearest-neighbor query processing algorithm for a database server using a spatial mashup to access driving time from a Web mapping service, e.g., Google Maps. We first designed two object grouping methods (i.e., MinIn and NearestIn) to group objects to intersections based on the road network topology to achieve shared execution and use a pruning technique to reduce the number of expensive external requests to the Web mapping service. We also extended our algorithm to support one-way road segments, and designed a user grouping method for shared execution to further reduce the number of external requests. We compare the performance of our algorithms with the basic algorithm through simulations and experiments. The results show that our algorithms significantly reduce the number of external requests and provide highly accurate of k-NN query answers.

References

[1] Jensen, C.S.: Database aspects of location-based services. In: Location-Based Services. Morgan Kaufmann (2004) 115–148

[2] Lee, D.L., Zhu, M., Hu, H.: When location-based services meet databases. Mobile Information Systems **1**(2) (2005) 81–90

[3] Gedik, B., Liu, L.: MobiEyes: A distributed location monitoring service using moving location queries. IEEE TMC **5**(10) (2006) 1384–1402

[4] Mokbel, M.F., Xiong, X., Aref, W.G.: SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In: ACM SIGMOD. (2004)

[5] Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB. (2003)

[6] Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: ACM SIGMOD. (2005)

[7] Mouratidis, K., Papadias, D., Hadjieleftheriou, M.: Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: ACM SIGMOD. (2005)

[8] Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: VLDB. (2002)

[9] Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Efficient k-nearest neighbor search in time-dependent spatial networks. In: DEXA. (2010)

[10] Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Towards k-nearest neighbor search in time-dependent spatial network databases. In: International Workshop on Databases in Networked Systems. (2010)

[11] George, B., Kim, S., Shekhar, S.: Spatio-temporal network databases and routing algorithms: A summary of results. In: SSTD. (2007)

[12] Vancea, A., Grossniklaus, M., Norrie, M.C.: Database-driven web mashups. In: IEEE ICWE. (2008)

[13] Levandoski, J.J., Mokbel, M.F., Khalefa, M.E.: Preference query evaluation over expensive attributes. In: Proc. of ACM CIKM. (2010)

[14] Google Maps/Google Earth APIs Terms of Service (Last updated: May 27, 2009): http://code.google.com/apis/maps/terms.html

[15] Google Maps: http://maps.google.com

[16] Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on a road network with speed patterns. In: IEEE ICDE. (2006)

[17] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Second Edition. The MIT Press (2001)

[18] Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: ACM SIGMOD. (2008)

[19] Huang, X., Jensen, C.S., Saltenis, S.: The islands approach to nearest neighbor querying in spatial networks. In: SSTD. (2005)

[20] Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: IEEE ICDE. (2002)

[21] Chang, K.C.C., Hwang, S.W.: Minimal probing: Supporting expensive predicates for top-k queries. In: ACM SIGMOD. (2002)