Using Parallel Spatial Mashup Model to Process k-NN Queries

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

[1]University of Science and Technology of China, Hefei, China
[2]City University of Hong Kong, Hong Kong, China
tianzdt@mail.ustc.edu.cn {chiychow,itqli}@cityu.edu.hk {xinming,ylxu}@ustc.edu.cn

Abstract. In this paper, we focus on k-NN queries where the distance between two locations is measured in terms of travel time. Generally speaking, it is costly for an LBS provider to collect real-time traffic data from vehicles or roadside sensors to compute the travel time and route information between two locations. Thus, we design a server-side spatial mashup model that enables an LBS provider to efficiently evaluate k-NN queries using the route information and travel time retrieved from an open/external Web mapping service, e.g., Google Maps, Microsoft Bing Maps, MapQuest Maps, Yahoo! Maps, Baidu Maps and so on. Due to the relative high cost and limitations of retrieving such external information, we take advantage of pruning, direction sharing and parallel requesting to reduce the number of external requests and query response time. We evaluate the proposed algorithms using MapQuest Maps, a real road network, real and synthetic data sets, and experimental results demonstrate the efficiency and scalability of our proposed algorithms.

1 Introduction

With the development of Web services and cloud computing, mashups which combines data, representation, and/or functionality from multiple Web applications to create a new application [1], attract more and more attentions from industry and academia. Typical types of mashups include spatial (i.e., mapping or GIS), search, social and photo. Based on the latest statistics of Programmable Web [2], the spatial mashup is the most popular type, which is built on Web mapping services, e.g., Google Maps, Microsoft Bing Maps, MapQuest Maps, Yahoo! Maps, Baidu Maps and so on.

On the other hand, with the ubiquity of wireless Internet access, GPS-enabled mobile devices and the advance in spatial database management systems, location-based services (LBS) which provide valuable information to their users based on their locations are widely used in people's daily lives (e.g., find the nearest gas stations to my car). However, the distance measure of spatial queries such as k-nearest-neighbor and range queries in LBS is mainly based on the network distance [3] instead of travel time in a road network, which would hide the fact that the user may take longer time to travel to his/her nearest object of interest (e.g., a restaurant and hotel) than other ones due to many realistic factors including heterogeneous traffic conditions and traffic accidents.

Meanwhile, travel time is highly dynamic [4], and it is difficult for every LBS provider to do real-time travel time computation/estimation due to the very expensive deployment cost [5, 6].

In this paper, we combine the advantage of Web mapping services, which can provide travel time and direction information based on current traffic conditions, and the desire of LBS providers, who want to provide users with query results in terms of travel time, to design a framework that uses a server-side spatial mashup model to process k-nearest-neighbor (k-NN) queries. The application scenario is that LBS providers can subscribe travel time and detailed route information from Web mapping services through their APIs to provide services for their users based on current traffic conditions. Because accessing external data is much more expensive than local data [7], and due to the limitations from Web mapping service providers [5, 6], we propose pruning, direction sharing and parallel requesting to reduce the number of external requests to the Web mapping service and query response time to users.

2 Related Work

Our focus in this paper is on k-NN queries in time-dependent road networks, where the cost or weight of road segments can change with time. Our work distinguishes itself from previous work [4, 8, 9] in that it does not model the underlying road network based on different criteria. Instead, it employs third party Web mapping services, e.g., MapQuest Maps, to compute the travel time of road networks and provide the route and travel time information through server-side spatial mashups.

There are some query processing algorithms designed to deal with expensive attributes that are accessed from external Web services (e.g., [5–7, 10, 11]). However, all previous works aim at minimizing the number of external requests, while ignore how to reduce the query response time to users with parallel requesting to third party services. Even though parallel processing has been widely used in spatio-temporal database, existing works mainly focus on parallel spatial joins over local data instead of external data, like parallel spatial similarity joins based on spatial and textual attributes [12], data partitioning for parallel spatial join processing [13], and parallel non-blocking spatial join algorithm [14], which are totally different from our work as to be presented subsequently.

3 System Model

In this section, we present our spatial mashup model, road network model and problem definition.

Spatial mashup model. Figure 1 depicts the spatial mashup model 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 an LBS provider through their mobile devices at anywhere, anytime. The database server processes queries based on local data, e.g., the location and basic information of

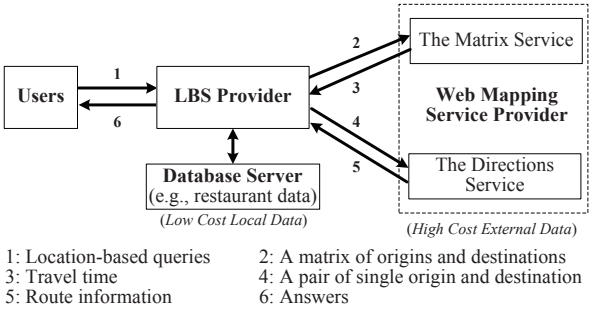


Fig. 1. Spatial Mashup Model.

restaurants, and external data accessed from a Web mapping service provider, i.e., travel time and detailed route information.

A typical Web mapping service provider offers two kinds of services, i.e., the Matrix service (e.g., the Google Distance Matrix API [15] and the MapQuest Route Matrix Service [16]) and the Directions service (e.g., the Google Directions API [17] and the MapQuest Directions Web Service [18]). The Matrix service provides travel distance and time for a matrix of origins and destinations. For example, given a request with a set of start points $\mathcal{S} = \{S_1...S_m\}$ and destination points $\mathcal{D} = \{D_1...D_n\}$, the service returns travel time $Time(S_i \rightarrow D_j)$ and travel distance $Dist(S_i \rightarrow D_j)$ ($S_i \in \mathcal{S}$ and $D_i \in \mathcal{D}$) for each pair in one response. By utilizing the Matrix service, the LBS provider can compute k-NN query answers based on travel time (Steps 2 and 3 as depicted in Figure 1). However, this kind of service does not return detailed route information. Route information can be obtained by passing the desired single origin and destination to the Directions service [15, 16]. Therefore, for each querying user, the LBS provider still needs to issue external requests to the Directions service to get detailed route information from the user to his corresponding k nearest neighbors (Steps 4 and 5 in Figure 1).

Road network model. A road map is modeled as a graph $G = (V, E)$ comprising a set V of vertices with a set E of edges, where each road segment is an edge and each intersection of road segments is a vertex.

Problem definition. Given a set of objects $\mathcal{R}$ and a k-NN query $Q = (\lambda, k)$ from a user U, where λ is U's location, and k is the maximum number of returned objects, the LBS provider returns U at most k objects in $\mathcal{R}$ with the shortest travel time from λ based on the route and travel time information accessed from a Web mapping service provider. Since the access to the Web mapping service is expensive, our objectives are to reduce the number of external Web mapping requests and user query response time through proposed optimizations.

4 Query Answer Computation Using the Matrix Service

The Matrix service can provide travel distance and time for a matrix of origins and destinations. However, each query sent to the Matrix service is usually limited by the number of allowed elements, where the number of origins times the

number of destinations defines the number of elements. For example, the Google Distance Matrix service only allows 100 elements per query, 100 elements per 10 seconds and 2,500 elements per 24 hour period for evaluation users [15]. Furthermore, there could be a very large number of objects in the spatial data set $\mathcal{R}$. It is extremely inefficient to access travel time from the user to all objects in $\mathcal{R}$ by issuing external requests to the Matrix service to compute a k-NN query answer. To this end, we employ the existing incremental network expansion(INE) algorithm in a road network [3] and existing pruning techniques designed for query processing with external data [5–7, 10, 11] to minimize the number of external Web mapping requests.

Given a road network model $G = (V, E)$, a set of objects $\mathcal{R}$, a k-NN query $Q = (\lambda, k)$ from a user U, the maximum movement speed V_{max} restricted by the user or the road network, and the number of allowed elements n defined by the Matrix service provider, our k-NN query answer computation algorithm has two major steps: (1) ***Destination selection step.*** This step first finds the n nearest objects based on their network distance by INE algorithm [3] from the U to the objects in $\mathcal{R}$ as the query destination set $\mathcal{D}$, or take $\mathcal{R}$ as $\mathcal{D}$ if the number of objects in $\mathcal{R}$ is less than n. The algorithm issues an external request to the Matrix service to retrieve the travel time from U to each object R_i in $\mathcal{D}$, i.e., $Time(U \rightarrow R_i)(R_i \in \mathcal{D})$. Based on the travel time information retrieved from the Matrix service, this step updates a current answer $\mathcal{A}$ by selecting the k objects with the shortest travel time from U. All the processed objects are removed from $\mathcal{R}$ and $\mathcal{D}$ is set to empty. (2) ***Pruning step.*** This step calculates the maximum possible network distance $dist_{max}$ by multiplying the largest travel time T_{max} of the objects in $\mathcal{A}$ with the maximum movement speed V_{max} to prune the unprocessed objects in $\mathcal{R}$ that are definitely not part of the query answer. If $\mathcal{R}$ is not empty or the current network expansion distance of INE is smaller than $dist_{max}$, the *destination selection step* is executed again; otherwise, $\mathcal{A}$ is returned to U as a query answer.

5 Route Information Access Using the Direction Service

Since the Matrix service only provides travel distance and time for a matrix of origins and destinations, the database server still needs to issue external requests to the Directions service to access detailed route information from a user to his k-nearest neighbors. A naive approach is simply passing the user's location λ and the location of each object R_i in $\mathcal{A}$ (computed by our k-NN query answer processing algorithm in Section 4) to the Directions service to retrieve detailed route information from U to R_i, i.e., $Route(U \rightarrow R_i)$. Using the naive approach, k external requests are needed. However, if k is large and/or there are a large number of querying users, the database server needs to issue a huge amount of requests to the Directions service, which may lead a high response time to users too. In this section, we introduce a direction sharing optimization (Section 5.1) to minimize the number of external requests to the Directions service, and parallel requesting (Section 5.2) to reduce query response time.

Algorithm 1 Accessing route information with direction sharing

1: **input:** $G = (V, E)$, $\mathcal{A}$
2: **while** $\mathcal{A}$ is not empty **do**
3: $R_l \leftarrow$ the object has the largest travel time in $\mathcal{A}$
4: Issue an external request to the Directions service to retrieve $Route(U \rightarrow R_l)$
5: Remove R_l and the shared object(s) between $\mathcal{A}$ and $Route(U \rightarrow R_l)$ from $\mathcal{A}$
6: **end while**
7: Return detailed route information accessed

5.1 Direction Sharing

For a request from location A to location B, the Directions service will return the route $Route(A \rightarrow B)$ with the shortest travel time, the turn-by-turn route and travel time information. Let X be an intermediate point in $Route(A \rightarrow B)$, i.e., $T = Route(A \ldots \rightarrow X \rightarrow \ldots B)$. We have proven the prefix property that every prefix in T, i.e., $Route(A \rightarrow X)$, is also the route with the shortest travel time from A to X [6]. Thus, the route information from A to X can be obtained from the route from A to B. In other words, $Route(A \rightarrow X)$ can share the direction information with $Route(A \rightarrow B)$, so no request is needed to access $Route(A \rightarrow X)$.

Algorithm. To get the detailed route information from a user U to each object in his query answer set $\mathcal{A}$ with the least number of external requests to the Directions service, the optimal algorithm should first process the object $R_h \in \mathcal{A}$ such that the direction information from U to R_h can be shared with the largest subset $\mathcal{A}'$ of other objects in $\mathcal{A}$ (i.e., R_h has the highest *sharing ability*). Since we do not know the detailed route information from U to any object in $\mathcal{A}$ until we issue external requests to the Directions service, it would be impossible to calculate *sharing ability* previously.

However, since we know the travel time from U to each object in $\mathcal{A}$, the object R_l which has the largest travel time in $\mathcal{A}$ is impossible to be located in any other route, i.e, $Route(U \rightarrow R_l) \not\subset Route(U \rightarrow R_i)(R_i \in \mathcal{A}, R_i \neq R_l)$. That is to say, an external request from U to R_l is inevitable no matter R_l has the highest *sharing ability* or not. Based on this point and the idea of direction sharing, Algorithm 1 is designed to access detailed route information from U to each object in $\mathcal{A}$.

5.2 Parallel Requesting

To reduce user query response time, an algorithm with parallel requesting is designed in this section by issuing requests to the Directions service in parallel instead of one by one. As discussed in Section 5.1, the route from U to an object in his query answer set $\mathcal{A}$ may be shared by the route(s) from U to other object(s) in $\mathcal{A}$. To ensure each request in each round of requesting is not wasteful, the returned routes from the Directions service should be independent, i.e., they should not be the sub-route of each other.

Similarly, since we do not know the detailed route information from U to any object in $\mathcal{A}$, it is impossible to precisely determine which objects can be processed in parallel by issuing external requests from U to them at the same time. Nevertheless, as we know the travel time from U to each object in $\mathcal{A}$

and can calculate the shortest network distance between any two points in the given road network $G = (V, E)$, we can determine whether two given routes are independent of each other or not based on following theorem:

Theorem 1. *Given the travel time of the routes from point A to points B and C in a road network, i.e., $Time(A \rightarrow B)$ and $Time(A \rightarrow C)$ ($Time(A \rightarrow B) < Time(A \rightarrow C)$, or vice versa), the shortest network distance $NDist(B \rightarrow C)$ from B to C, and the maximum movement speed V_{max}, the route $Route(A \rightarrow B)$ from A to B is independent of the route $Route(A \rightarrow C)$ from A to C if the following inequality holds:*

$$Time(A \rightarrow B) + \frac{NDist(B \rightarrow C)}{V_{\max}} > Time(A \rightarrow C) \tag{1}$$

Proof. If $Route(A \rightarrow B)$ is a sub-route of $Route(A \rightarrow C)$, then

$$Time(A \rightarrow C) = Time(A \rightarrow B) + Time(B \rightarrow C),$$

where $Time(B \rightarrow C)$ is the travel time of the route from B to C. As $NDist(B \rightarrow C)$ and V_{max} stand for the shortest network distance between B and C and the maximum movement speed, respectively, $\frac{NDist(B \rightarrow C)}{V_{\max}}$ indicates the minimum possible travel time from B to C, i.e.,

$$\frac{NDist(B \rightarrow C)}{V_{\max}} \leq Time(B \rightarrow C).$$

Thus,

$$Time(A \rightarrow B) + \frac{NDist(B \rightarrow C)}{V_{\max}} \leq Time(A \rightarrow C),$$

which is contradictory to the assumption. □

Theorem 1 gives theoretical basis for selecting objects from $\mathcal{A}$ which can be processed in parallel, i.e., if $Time(U \rightarrow R_i) + \frac{NDist(R_i \rightarrow R_j)}{V_{\max}} > Time(U \rightarrow R_j)$ ($Time(U \rightarrow R_i) < Time(U \rightarrow R_j)$), the route information from U to R_i and R_j can be retrieved at the same time.

Algorithm. Given a road network model $G = (V, E)$, the query answer $\mathcal{A}$ of $Q = (\lambda, k)$ from a user U, and the maximum movement speed $V_{\max}$, Algorithm 2 depicts the pseudo code for accessing detailed route information from U to objects in $\mathcal{A}$ by issuing external requests to the Directions service in parallel. For each round of parallel requesting, Algorithm 2 has two main steps: (1) **Destination set finding step.** Initially, the destination set $\mathcal{A}_p$ is set to empty to store the objects, to which the database server can send external requests from U at the same time for this round. As the object with the largest travel time in $\mathcal{A}$ is impossible to be located in any other returned routes as presented in Section 5.1, the algorithm removes it from $\mathcal{A}$ and inserts it into $\mathcal{A}_p$ (Lines 3 to 4 in Algorithm 2). Then, starting from the object with the largest travel time, each object R_l remained in $\mathcal{A}$ is checked with each object R_k in $\mathcal{A}_p$ based on

Algorithm 2 Accessing route information with parallel requesting

```
1: input: G = (V, E), A, V_max
2: while A is not empty do
3:     A_p ← ∅
4:     Remove the object with the largest travel time from A and insert it into A_p
5:     A' ← A
6:     while A' is not empty do
7:         R_l ← the object with the largest travel time in A'
8:         for each object R_k in A_p do
9:             if Time(U → R_l) + NDist(R_l → R_k) / V_max is not larger than Time(U → R_k) then
10:                Break
11:            end if
12:            if R_k is the last object in A_p then
13:                Insert R_l into A_p and remove it from A
14:            end if
15:        end for
16:        Remove R_l from A'
17:    end while
18:    Issue requests to the Directions service in parallel to retrieve Route(U → R_i), R_i ∈ A_p
19:    for Each object R_i in A_p do
20:        A ← A− objects in Route(U → R_i)
21:    end for
22: end while
23: Return detailed route information accessed
```

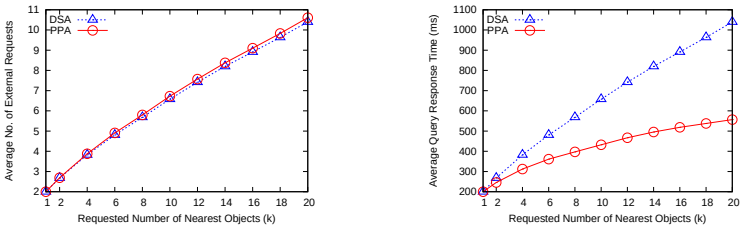
Theorem 1 to determine whether R_l can be safely processed in this round. Only if R_l satisfies the condition of Theorem 1 (i.e., Inequation 1) with each object in $\mathcal{A}_p$, R_l can be taken as one of the parallel requesting objects (Lines 5 to 17 in Algorithm 2). (2) **Parallel requesting step.** After having found the destination set $\mathcal{A}_p$, external requests to the Directions service are issued in parallel to retrieve the route information from U to all objects in $\mathcal{A}_p$ at the same time (Line 18 in Algorithm 2). Based on the prefix property, objects in each returned route are removed from $\mathcal{A}$ (Lines 19 to 21 in Algorithm 2).

The algorithm continues to execute the above two steps for the next round until $\mathcal{A}$ becomes empty. By that time, the detailed route information from U to each object in $\mathcal{A}$ has been retrieved. The worst case of Algorithm 2 is that only one object is selected for each round of parallel requesting. In this situation, user response time by Algorithm 2 is the same as that by Algorithm 1.

6 Experimental Evaluation

6.1 Evaluation Model

Evaluated approaches. For each evaluated approach, our k-NN query answer computation algorithm (described in Section 4) is the first one to be executed through the Matrix service. Then, the following algorithms are conducted to access the detailed route information from each querying user to each of his k-nearest objects through the Directions service. (1) Baseline algorithm (denoted as DSA): Since direction sharing optimization has been proposed in [6], we take the algorithm of accessing route information with direction sharing (Algorithm 1) as the baseline algorithm. (2) The proposed algorithm of accessing



(a) No. of external requests (b) Query response time

Fig. 2. Effect of the value of k.

route information with direction sharing and parallel requesting (Algorithm 2, denoted as PPA).

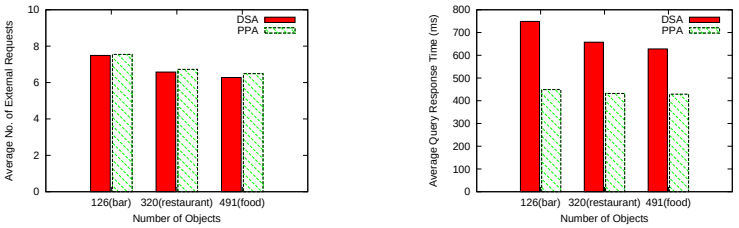
Performance metrics. We evaluate the performance in terms of two metrics: (1) the average number of external Web mapping requests per user query, including the number of requests to both the Matrix service and the Directions service, and (2) the average response time per user query, which is defined as the finished time subtracting the arrival time of the query.

Experiment settings. We use C++ and MapQuest Maps with a real road network of Hennepin County, MN, USA [19]. We select an 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. We use three real data sets within the selected area, i.e., a bar data set of 126 bars, a restaurant data set of 320 restaurants and a food data set of 491 food places. Unless mentioned otherwise, we use the real restaurant data set, and uniformly generate 10,000 querying users in the road network. The default k value is 10, and the maximum allowed speed is 120 km per hour.

6.2 Experimental Result Analysis

Effect of the Value k Figure 2 shows the performance of the baseline approach DSA and the advanced approach PPA with respect to various numbers of nearest objects (i.e., k) from 1 to 20. When k gets larger, there are more objects in the answer set of a querying user, and thus the database server needs to issue more external requests, which induces a higher response time. Compared to DSA, PPA almost yields the same average number of external Web mapping requests (Figure 2a), while reduces the query response time by 35.6% on average (Figure 2b). The results indicate that PPA can largely reduce the average query response time to users without increasing the number of external requests because of parallel requesting as presented in Section 5.2.

Effect of the Number of Objects Figure 3 shows the performance of DSA and PPA with respect to three real data sets, i.e., the bar data set, the restaurant data set and the food data set. Similarly, PPA can reduce the query response time by 35.7% while keep the almost same number of external Web mapping requests



(a) No. of external requests (b) Query response time

Fig. 3. Effect of the number of objects.

as DSA. Along with the size increase of the data sets, both the average number of external Web mapping requests (Figure 3a) and the average query response time (Figure 3b) of DSA and PPA are decreased. This is because the larger data set the higher object density in the given road network, and hence a returned route from the Web mapping service can be shared by more objects. Thus, the number of external Web mapping requests per querying user is reduced, which yields the decrease of the query response time accordingly. The results of this experiment confirm that our proposed direction sharing and parallel requesting optimizations can scale up to a higher object density with better performance.

7 Conclusion

This paper has presented a server-side spatial mashup model by combining the Matrix service and the Directions service together. Our model enables an LBS provider to efficiently evaluate k-NN queries using the route information and travel time accessible from an external Web mapping service, e.g., Google Maps. We have taken the advantage of existing incremental network expansion and pruning techniques to compute the k-NN query answer based on the Matrix service, then designed direction sharing and parallel requesting algorithm to reduce the number of external requests to the Directions service and query response time. To evaluate the performance of our proposed algorithms, we have conducted extensive experiments using MapQuest Maps, a real road network, three real object data sets, and a synthetic user data set. The experiment results show that our proposed algorithms significantly reduce the number of external requests and user query response time.

8 Acknowledgements

The work presented in this paper was partially supported by a multi-server project [20] [21] and a multi-resolution data management project [22] [23].

References

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

[2] ProgrammableWeb: http://www.programmableweb.com
[3] Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB. (2003)
[4] Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Efficient k-nearest neighbor search in time-dependent spatial networks. In: DEXA. (2010)
[5] Zhang, D., Chow, C.Y., Li, Q., Zhang, X., Xu, Y.: Efficient evaluation of k-NN queries using spatial mashups. In: SSTD. (2011)
[6] Zhang, D., Chow, C.Y., Li, Q., Zhang, X., Xu, Y.: SMashQ: Spatial mashup framework for k-NN queries in time-dependent road networks. Distributed and Parallel Databases **31**(2) (2013) 259–287
[7] Levandoski, J.J., Mokbel, M.F., Khalefa, M.E.: Preference query evaluation over expensive attributes. In: CIKM. (2010)
[8] Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Towards k-nearest neighbor search in time-dependent spatial network databases. In: DNIS. (2010)
[9] George, B., Kim, S., Shekhar, S.: Spatio-temporal network databases and routing algorithms: A summary of results. In: SSTD. (2007)
[10] Bruno, N., Gravano, L., Marian, A.: Evaluating top-k queries over web-accessible databases. In: ICDE. (2002)
[11] Chang, K.C.C., Hwang, S.W.: Minimal probing: Supporting expensive predicates for top-k queries. In: SIGMOD. (2002)
[12] Ballesteros, J., Cary, A., Rishe, N.: SpSJoin: Parallel spatial similarity joins. In: GIS. (2011)
[13] Zhou, X., Abel, D.J., Truffet, D.: Data partitioning for parallel spatial join processing. GeoInformatica **2** (1998) 175–204
[14] Luo, G., Naughton, J., Ellmann, C.: A non-blocking parallel spatial join algorithm. In: ICDE. (2002)
[15] The Google Distance Matrix API: https://developers.google.com/maps/documentation /distancematrix
[16] MapQuest Directions Web Service - Route Matrix: http://www.mapquestapi.com/directions/#matrix
[17] The Google Directions API: https://developers.google.com/maps/documentation /directions
[18] MapQuest Directions Web Service: http://www.mapquestapi.com/directions
[19] TIGER/Line Shapefiles 2009 for: Hennepin County, Minnesota: http://www2.census.gov/cgi-bin/shapefiles2009/county-files?county=27053
[20] Ng, B., Li, F., Lau, R., Si, A., Siu, A.: A performance study of multi-server systems for distributed virtual environments. Information Sciences **154**(1) (2003) 85–93
[21] Ng, B., Lau, R., Si, A., Li, F.: Multi-server support for large scale distributed virtual environments. IEEE Trans. on Multimedia **7**(6) (2005) 1054–1065
[22] To, D., Lau, R., Green, M.: A method for progressive and selective transmission of multi-resolution models. In: Proc. ACM VRST. (1999) 88–95
[23] To, D., Lau, R., Green, M.: An adaptive multi-resolution method for progressive model transmission. Presence: Teleoperators and Virtual Environments **10**(1) (2001) 62–74