# A Spatial Mashup Service for Efficient Evaluation of Concurrent $k$-NN Queries

Detian Zhang, Chi-Yin Chow, Qing Li, Xinming Zhang and Yinlong Xu

**Abstract**—Although the travel time is the most important information in road networks, many spatial queries, e.g., $k$-nearest-neighbor ($k$-NN) and range queries, for location-based services (LBS) are only based on the network distance. This is because it is costly for an LBS provider to collect real-time traffic data from vehicles or roadside sensors to compute the travel time between two locations. With the advance of web mapping services, e.g., Google Maps, Microsoft Bing Maps, and MapQuest Maps, there is an invaluable opportunity for using such services for processing spatial queries based on the travel time. In this paper, we propose a server-side **S**patial **M**ashup **S**ervice (SMS) that enables the LBS provider to efficiently evaluate $k$-NN queries in road networks using the route information and travel time retrieved from an external web mapping service. Due to the high cost of retrieving such external information, the usage limits of web mapping services, and the large number of spatial queries, we optimize the SMS for a large number of $k$-NN queries. We first discuss how the SMS processes a single $k$-NN query using two optimizations, namely, *direction sharing* and *parallel requesting*. Then, we extend them to process multiple concurrent $k$-NN queries and design a performance tuning tool to provide a trade-off between the query response time and the number of external requests and more importantly, to prevent a starvation problem in the parallel requesting optimization for concurrent queries. We evaluate the performance of the proposed SMS using MapQuest Maps, a real road network, real and synthetic data sets. Experimental results show the efficiency and scalability of our optimizations designed for the SMS.

**Index Terms**—Spatial mashups, location-based services, $k$-NN queries, web mapping services, road networks

## 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 the industry [2], [3] and the research community [4]–[7]. Typical types of mashups include spatial (i.e., mapping or GIS), search, social and photo. Based on the latest statistics of Programmable Web [8], the spatial mashup is the most popular type, which is built on web mapping services, e.g., Google Maps [9], MapQuest Maps [10], Microsoft Bing Maps [11], Yahoo! Maps [12], and Baidu Maps [13]. However, most of existing spatial mashups are very simple, such as displaying points of interest on a web map. Nevertheless, existing web mapping service providers can provide services far more than such simple kind of operations. For example, within one response, Google Maps [9] can provide travel time and detailed route information between two locations and even travel distance and time for a matrix of origins and destinations based on real-time traffic conditions.

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 [14], [15] 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 ($k$-NN) and range queries in LBS is mainly based on the network distance [16] instead of travel time in a road network, which would hide the fact that the user may take longer time to travel to her/his nearest object of interest (e.g., a restaurant and a hotel) than other ones due to many realistic factors including heterogeneous traffic conditions and traffic accidents [17]–[19].

Travel 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 [20]. Hence, it is almost impossible to accurately predict the travel time between two locations in the road network based on their network distance. The best way to provide real-time travel time computation is to continuously monitor the traffic in the road network. However, it is difficult for every LBS provider to do so due to the expensive setup cost.

In this paper, we design a server-side Spatial Mashup Service (SMS) to utilize web mapping services, which can provide travel time and direction information based on current traffic conditions, to process $k$-NN queries in terms of travel time in road networks. Web mapping service providers have many sources to collect data (e.g., historical traffic statistics and real-time traffic conditions) to estimate/calculate the travel time and direction information between any two locations. Our proposed SMS provides a cost-effective way for a database server to access the route and travel time information between two locations in a road network from external web mapping services. The application scenario of this work is that LBS providers can subscribe travel time and detailed route information from web mapping services through their APIs to provide

• *D. Zhang is with the School of Digital Media, Jiangnan University, Wuxi, China. C.-Y. Chow and Q. Li are with the Department of Computer Science, City University of Hong Kong, Hong Kong, China. X. Zhang and Y. Xu are with the Department of Computer Science, University of Science and Technology of China, Hefei, China. E-mail: tianzdt@mail.ustc.edu.cn, {chiychow, itqli}@cityu.edu.hk, {xinming, ylxu}@ustc.edu.cn.*

services for their users based on current traffic conditions. In other words, our SMS enables LBS providers to outsource the real-time traffic monitoring and traffic analysis operations to web mapping service providers, and thus, an LBS provider can focus on its core business activities and reduce its operational cost.

The key research challenges of this work are threefold. (1) Accessing external data is much more expensive than local data. For example, retrieving travel time from the Microsoft 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 [21]. (2) The usage of web mapping service providers is limited [22], [23], e.g., the maximum limits of requests and the restricted use of retrieved route information. (3) An LBS system has to deal with a larger number of concurrent queries from mobile users.

To address these challenges, in this work, we first propose two optimizations, namely, *direction sharing* and *parallel requesting*, using the web mapping matrix and directions services, for a single $k$-NN query based on the travel time. To further improve system performance, we extend these two optimizations for a number of concurrent $k$-NN queries. In addition, we design a performance tuning tool to provide a trade-off between the query response time and the number of external requests, and more importantly, it can prevent a starvation problem in the parallel requesting optimization for concurrent queries.

In general, the main contributions of our work in this paper can be summarized as follows:

- We design a server-side Spatial Mashup Service (SMS) and propose a basic $k$-NN query processing algorithm using the matrix service to compute a query answer based on travel time in road networks. (Section 3)
- We propose a parallel requesting optimization for processing external web mapping requests to reduce the query response time of a single $k$-NN query. (Section 4)
- We extend the parallel requesting optimization for multiple concurrent queries, and introduce a tuning tool to balance between the query response time and the number of external requests, and prevent a starvation problem in the parallel requesting optimization. (Section 5)
- We conduct extensive experiments to evaluate the performance of the proposed optimizations for our SMS. (Section 6)

The rest of this paper is organized as follows. Section 2 highlights related work. Section 3 describes the system model and the basic $k$-NN query processing algorithm. Sections 4 and 5 present the parallel requesting optimization for a single query and multiple concurrent queries, respectively. Experimental results are analyzed in Section 6. Finally, we conclude this paper in Section 7.

## 2 Related Work

In this section, we highlight the related work of our framework.

**Location-based queries in road networks.** Existing location-based query processing algorithms in road networks can be categorized into two main models. (1) ***Time-independent road networks.*** In this model, 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., [16], [24], [25]). 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 time of a day due to dynamic traffic on road segments [20], [26]. (2) ***Time-dependent road networks.*** The location-based query processing algorithms designed for this model have the ability to support dynamic weights of road segments and topology of a road network, which can change with time. This model is more realistic but more challenging. George et al. [27] proposed a time-aggregated graph, which uses time series to represent time-varying attributes. In time-dependent road networks where the weight of each road segment is a function of time, Demiryurek et al. [20], [26] and Ding et al. [28] proposed solutions for processing $k$-NN queries and the time-dependent shortest-path problem, respectively.

**External web services for location-based queries.** Some query processing algorithms are designed to deal with expensive attributes that are accessed from external web services (e.g., [21]–[23], [29]–[31]). To minimize the number of external requests, [21], [29], [30] mainly focused on using either some *cheap* attributes that can be retrieved from local data sources [21], [29] or sampling methods [30] to prune a whole set of objects into a smaller set of candidate objects; also they only issue necessary external requests for retrieving candidate objects.

**Mapping services for location-based queries.** There are also a few studies related to mapping services for location-based queries [21], [32], but none of them aims at processing location-based queries based on the direction information retrieved from web mapping services in a real time manner, except for our previous work [22], [23], [31]. For example, in [21], the authors explored an efficient framework for processing skyline and multi-objective queries in a database when the data involves a mix of "cheap" and "expensive" attributes, where driving time retrieved from mapping services is only one of "expensive" attributes, instead of utilizing the direction information for query processing as we used in our work; in [32], the authors designed a TAREEG system, which employs MapReduce-based techniques to extract the whole static map data (e.g., a road network) of an area from OpenStreetMap into local databases for researchers or individuals.

Our previous work [22], [23], [31] proposed spatial query processing algorithms that use grouping and direction sharing optimizations based on the road network topology, shared query execution, and pruning techniques to reduce the number of external requests and provide highly accurate query answers. As far as we know, we are the first one to design such a server-side spatial mashup system that utilizes the direction information retrieved from web mapping services for spatial query processing.

**Parallel processing for location-based queries.** Although parallel query processing has been widely used in

spatio-temporal databases, most of existing approaches only focus on parallel spatial joins over local data instead of external data, like parallel spatial similarity joins based on spatial and textual attributes [33], data partitioning for parallel spatial join processing [34], and parallel non-blocking spatial join algorithm [35]. These approaches cannot be applied to our framework because it aims at processing parallel requests for external web mapping services.

This paper focuses on $k$-NN queries in time-dependent road networks. Our work distinguishes itself from previous work [20], [26], [27] in that it does not model the underlying road network based on different criteria. Instead, it employs external web mapping services, e.g., MapQuest Maps, to provide the route and travel time information in a road network through server-side spatial mashups by combing the matrix service and the directions service together, which is different from [22], [23] that only use the directions service. Besides, we consider multiple concurrent $k$-NN query processing in this paper instead of only single $k$-NN query processing as in [31].

Since the use of external web mapping requests is more expensive than accessing local data [21], we extend the direction sharing optimization and propose the parallel requesting optimization for a single $k$-NN query and multiple concurrent $k$-NN queries to reduce the number of external requests and the query response time.

## 3 System Model

In this section, we present our system architecture, road network model, problem definition, and basic $k$-NN query processing algorithm using spatial mashups.

**System architecture.** Figure 1 depicts the system architecture that consists of three entities: users, a database server at an LBS provider, and a web mapping service provider. Users send $k$-NN queries to the database server through their mobile devices at anywhere, anytime. The database server processes queries based on local data, e.g., the location and basic information of 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 [36] and the MapQuest Route Matrix Service [37]) and the **directions service** (e.g., the Google Directions API [38] and the MapQuest Directions Web Service [39]). 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 $\mathcal{S} = \{S_1, \ldots, S_m\}$ and destination points $\mathcal{D} = \{D_1, \ldots, D_n\}$, 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 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 [36], [37]. 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).

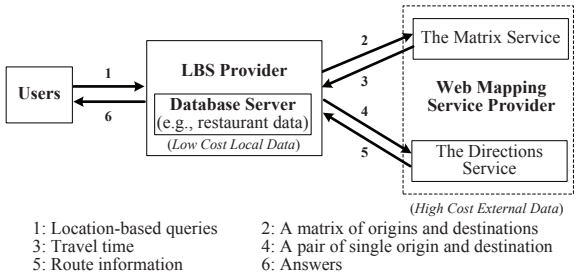



Fig. 1. System architecture.

**Road network model.** A road map, which can be extracted from OpenStreetMap [32], is modeled as a graph $G = (V, E)$ comprising a set of vertices $V$ with a set of edges $E$, where each road segment is an edge and each intersection of road segments is a vertex. For instance, Figure 2a depicts a real road map that is modeled as an undirected graph [1] (Figure 2b), where an edge represents a road segment (e.g., $I_1I_2$ and $I_1I_5$) and a square represents an intersection of road segments (e.g., $I_1$ and $I_2$).

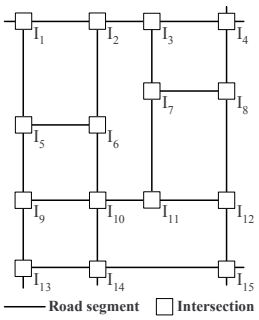
**Problem definition.** Given a set of objects $\mathcal{R}$ and a snapshot $k$-NN query $Q = (\lambda, k)$ from a user $U$, where $\lambda$ is $U$'s location, and $k$ is the maximum number of returned objects, the LBS provider returns $U$ at most $k$ objects in $\mathcal{R}$ with the shortest travel time from $\lambda$ 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 the query response time through our proposed optimizations and parallel requesting paradigm.

**Basic $k$-NN query processing algorithm using the matrix service.** The matrix service provides the travel distance and time for a matrix of origins and destinations. However, the matrix service usually has a limit on the number of elements, i.e., origins and destinations. For example, the Google Distance Matrix service only allows 100 elements per query, 100 elements per 10 seconds, and 2,500 elements per 24 hours for evaluation users [36]. Furthermore, since 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 [16] and existing pruning techniques designed for query processing with external data [21]–[23], [29], [30] 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 $V_{max}$ which can be defined by the user or restricted by the road network (i.e., the maximum legal

1. For simplicity, we assume that each road segment of the road network in our paper is bidirectional (modeled as an undirected graph). However, our proposed direction sharing and parallel requesting optimizations are also applicable to road networks where road segments are directional.


(a) A road map




(b) A graph model

Fig. 2. Road network model.

| $U_i$ | $Q_i$ | $A_i$ | Time($U_i \rightarrow R_j$) (seconds) |
|---|---|---|---|
| $U_1$ | $Q_1$ | $R_1$ | 50 |
| | | $R_2$ | 300 |
| | | $R_3$ | 550 |
| $U_2$ | $Q_2$ | $R_1$ | 200 |
| | | $R_2$ | 500 |
| | | $R_3$ | 700 |

Fig. 3. The example of query answers.

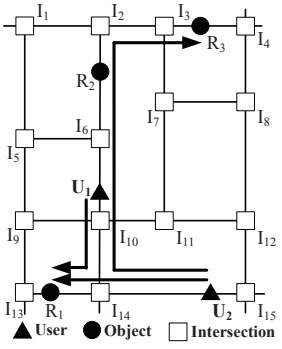



Fig. 4. The detailed route information.

speed in the road network), and the limited number of elements $n$ defined by the matrix service provider, our basic $k$-NN query processing algorithm has two major steps: (1) ***Destination selection step.*** This step first finds the $n$ nearest objects based on their network distance by the INE algorithm [16] from $U$ 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., $Time(U \rightarrow R_i)$. 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 $\mathcal{R}$. (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 $V_{max}$ (i.e., $dist_{max} = T_{max} \times V_{max}$) 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.

**Running example.** Suppose that $Q_1 = (\lambda_1, k_1 = 3)$ and $Q_2 = (\lambda_2, k_2 = 3)$, which are issued by users $U_1$ and $U_2$, respectively. The database server first receives $Q_1$, and then $Q_2$. Their query answers are computed separately by the basic $k$-NN query processing algorithm using the matrix service as shown in Figure 3. We use $Q_1$ and $Q_2$ in our running example.

**Algorithm 1** Direction sharing optimization

1: **input:** $G = (V, E)$, $\mathcal{A}$
2: **while** $\mathcal{A}$ is not empty **do**
3: $R_l \leftarrow$ the object having the longest 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 other objects with which $Route(U \rightarrow R_l)$ can be shared from $\mathcal{A}$
6: **end while**
7: Return the detailed route information for each object in $\mathcal{A}$

## 4 Parallel Requesting Optimization for a Single Query

Since the matrix service only provides the travel distance and time information for a matrix of origins and destinations, the database server still needs to issue external requests to the directions service to access the detailed route information from a user to each of her/his $k$-nearest objects (e.g., the user wants to know how to reach each object). A naive approach is simply passing the user's location $\lambda$ and the location of each object $R_i$ in $\mathcal{A}$ (computed by our basic $k$-NN query processing as described in Section 3) to the directions service to retrieve the detailed route information from $U$ to $R_i$, i.e., $Route(U \rightarrow R_i)$. Using this naive approach, $k$ external requests are needed. However, if $k$ is large and/or there are a large number of queries, the database server needs to issue a lot of requests to the directions service, which may lead to a long query response time and exceed the usage limits. In this section, we introduce a direction sharing optimization (Section 4.1) to minimize the number of external requests for the directions service, and a parallel request processing paradigm (Section 4.2) to reduce query response time.

### 4.1 Direction Sharing Optimization

For a request from location $A$ to location $B$, the directions service will return the route $Route(A \rightarrow B)$ with its shortest travel time and the turn-by-turn route and travel time information. Let $X$ be an intermediate location in $Route(A \rightarrow B)$, i.e., $T = Route(A \rightarrow \ldots \rightarrow X \rightarrow \ldots \rightarrow B)$. 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 $X$ [23]. Thus, the route information from $A$ to $X$ can be obtained from the route from $A$ to $B$. In other words, $Route(A \rightarrow B)$ can be shared with $Route(A \rightarrow X)$, so no request is needed to retrieve $Route(A \rightarrow X)$.

**Algorithm.** We observe that the object $R_l$ with the largest travel time is impossible to be located in the routes from $U$ to any other objects in $\mathcal{A}$, and the object with a longer travel time has a higher chance to share its route information with other objects. Based on these observations, Algorithm 1 is designed to access the detailed route information from $U$ to each object in $\mathcal{A}$ by first processing the object with the longest travel time.

**Example.** For $Q_1$ in our running example, since $R_3$ has the longest travel time in $\mathcal{A}_1$ (Figure 3), an external request is issued to the directions service to access the detailed route information from $U_1$ to $R_3$, i.e., $Route(U_1 \rightarrow R_3) = \{U_1 \rightarrow I_6 \rightarrow I_2 \rightarrow I_3 \rightarrow R_3\}$ as depicted in Figure 4, where users and objects are represented by triangles and

circles, respectively. As $R_2$ is located in the road segment $I_6I_2$, $Route(U_1 \rightarrow R_3)$ can be shared to find the route information from $U_1$ to $R_2$, i.e., $Route(U_1 \rightarrow R_2) = \{U_1 \rightarrow I_6 \rightarrow R_2\}$. The prefix property shows that $Route(U_1 \rightarrow R_2)$ gives the shortest travel time [23]. Therefore, $R_2$ and $R_3$ are removed from $\mathcal{A}_1$. Currently, only one object $R_1$ is in $\mathcal{A}_1$. Since $R_1$ is not located in $Route(U_1 \rightarrow R_3)$, an external request is issued to access the detailed information of $Route(U_1 \rightarrow R_1)$. As a result, instead of three external requests required by the naive approach, the direction sharing optimization (i.e., Algorithm 1) only needs two external requests.

Similarly, two external requests are needed to answer $Q_2$. In comparison with the naive approach, Algorithm 1 reduces the number of external requests required to answer $Q_1$ and $Q_2$ from six to four (i.e., 33.3%).

### 4.2 Parallel Requesting Optimization

Since web mapping services can process multiple directions requests in parallel, we design the parallel requesting optimization to issue external requests in parallel to further reduce the query response time [31]. For a user $U$'s $k$-NN query answer $\mathcal{A}$, a simple parallel requesting approach is issuing requests to the directions service for all $k$ objects in $\mathcal{A}$ at the same time. This simple approach can minimize the query response time, but it cannot make use of the direction sharing optimization as discussed in Section 4.1 (i.e., $k$ external requests are needed). To this end, we design the parallel requesting optimization to fulfill these two objectives (i.e., reducing the number of external requests and the query response time). The basic idea of this optimization is to issue external requests to directions service for the objects in $\mathcal{A}$ with independent routes (Definition 1) in parallel and process other objects as in the direction sharing optimization.

**Definition 1** (Independent routes)**.** *Two routes $Route(A \rightarrow B)$ and $Route(C \rightarrow D)$ are independent if they are not the sub-route of each other, i.e., $Route(A \rightarrow B) \not\subset Route(C \rightarrow D)$ and $Route(C \rightarrow D) \not\subset Route(A \rightarrow B)$.*

As we do not know the detailed route information from $U$ to any object in $\mathcal{A}$, so it is impossible to determine which objects can be processed in parallel. Nevertheless, since we know the travel time from $U$ to each object in $\mathcal{A}$ through the matrix service and calculate the shortest network distance between any two locations in the given road network $G = (V, E)$, we can use such information to find out some independent routes as shown in Theorem 4.1.

**Theorem 4.1.** *Given the travel time of the routes from location $A$ to locations $B$ and $C$ in a road network, i.e., $Time(A \rightarrow B)$ and $Time(A \rightarrow C)$, and $Time(A \rightarrow B) < Time(A \rightarrow C)$, the shortest network distance $NDist(B \rightarrow C)$ from $B$ to $C$, and the maximum movement speed $V_{max}$ defined by the user or restricted by the road network, the routes from $A$ to $B$ (i.e., $Route(A \rightarrow B)$) and from $A$ to $C$ (i.e., $Route(A \rightarrow C)$) are independent if the following inequality holds:*

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

*Proof.* If the inequality is true, $Route(A \rightarrow B)$ is not a sub-route of $Route(A \rightarrow C)$. Since $\frac{NDist(B \rightarrow C)}{V_{\max}}$ is the minimum possible travel time from $B$ to $C$, if $Route(A \rightarrow B)$ is a sub-route of $Route(A \rightarrow C)$, then

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

□

Theorem 4.1 gives theoretical basis for selecting objects from a user $U$'s query answer set $\mathcal{A}$ which can be processed in parallel. For any two objects $R_i$ and $R_j$ in $\mathcal{A}$, if $Time(U \rightarrow R_i)$ and $Time(U \rightarrow R_j)$ satisfy the inequality of the theorem, $Route(U \rightarrow R_i)$ and $Route(U \rightarrow R_j)$ can be retrieved through the directions service in parallel.

**Algorithm.** Algorithm 2 depicts the pseudo code for the direction sharing and parallel requesting optimizations with three inputs: a road network model $G = (V, E)$, the query answer $\mathcal{A}$ of $Q = (\lambda, k)$ for a user $U$, and the maximum movement speed $V_{\max}$. In general, it executes two main steps iteratively until $\mathcal{A}$ becomes empty.

(1) **Parallel destination set step.** A destination set of objects $\mathcal{A}_p$ is initially set to empty to store the objects that can be retrieved from the directions service in parallel. The objects in $\mathcal{A}$ are sorted with their travel time in decreasing order. 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 4.1, it is removed from $\mathcal{A}$ and inserted into $\mathcal{A}_p$ (Line 6 in Algorithm 2). For each object $R_l$ in $\mathcal{A}$, $R_l$ is checked with each object $R_k$ in $\mathcal{A}_p$ according to Theorem 4.1 to determine whether $R_l$ can be processed in parallel with the objects in $\mathcal{A}_p$. If $R_l$ satisfies the inequality of Theorem 4.1 with all other objects in $\mathcal{A}_p$, $R_l$ is added to $\mathcal{A}_p$ (Lines 7 to 19).

(2) **Parallel requesting step.** For all the objects in $\mathcal{A}_p$, external requests are issued in parallel to the directions service to retrieve their route information. Based on the prefix property, objects in each retrieved route are also removed from $\mathcal{A}$ (Lines 21 to 24). This is because the direction sharing optimization can use the retrieved route information to find the route from $U$ to each of these objects.

The algorithm keeps executing these two steps 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 *parallel destination set step*. In this situation, the query response time of Algorithm 2 is the same as that of Algorithm 1.

**Example.** The shortest network distance between every pair of objects in the running example is shown as in Figure 5a, and the maximum movement speed is 30 m/s, i.e., $V_{max} = 30$ m/s.

***For $Q_1$, in the first iteration.*** Its answer set is $\mathcal{A}_1 = \{R_3, R_2, R_1\}$, where the objects are sorted with their travel time in decreasing order (Figure 3). The *parallel destination set step* first sets $\mathcal{A}_p$ to empty. Since $R_3$ has the longest travel time in $\mathcal{A}_1$, $R_3$ is inserted into $\mathcal{A}_p$ and removed from $\mathcal{A}_1$, i.e., $\mathcal{A}_p = \{R_3\}$ and $\mathcal{A}_1 = \{R_2, R_1\}$. The *parallel destination set step* next processes $R_2$. Given $Time(U_1 \rightarrow R_2) = 300$ seconds, $NDist(R_2 \rightarrow R_3) = 5$ km (Figure 5a), and $V_{max} = 30$ m/s, we calculate that the shortest travel time from $U_1$ to $R_3$ via $R_2$ is $Time(U_1 \rightarrow R_2) + \frac{NDist(R_2 \rightarrow R_3)}{V_{max}} = 300 + \frac{5{,}000}{30} =$

**Algorithm 2** Direction sharing and parallel requesting optimizations

1: **input:** $G = (V, E)$, $\mathcal{A}$, $V_{max}$
2: **while** $\mathcal{A}$ is not empty **do**
3: *// Parallel destination set step*
4: $\mathcal{A}_p \leftarrow \{\}$
5: Sort the objects in $\mathcal{A}$ with their travel time in decreasing order
6: Move the object with the longest travel time from $\mathcal{A}$ to $\mathcal{A}_p$
7: **for** each object $R_l$ in $\mathcal{A}$ **do**
8: independent $\leftarrow$ true
9: **for** each object $R_k$ in $\mathcal{A}_p$ **do**
10: Compute $NDist(R_l \rightarrow R_k)$ based on $G = (V, E)$
11: **if** $Time(U \rightarrow R_l) + \frac{NDist(R_l \rightarrow R_k)}{V_{max}} \leq Time(U \rightarrow R_k)$ **then**
12: independent $\leftarrow$ false
13: **break**
14: **end if**
15: **end for**
16: **if** independent = true **then**
17: Insert $R_l$ into $\mathcal{A}_p$ and remove $R_l$ from $\mathcal{A}$
18: **end if**
19: **end for**
20: *// Parallel requesting step*
21: For all the objects in $\mathcal{A}_p$, external requests are issued in parallel to the directions service to retrieve their route information
22: **for** each retrieved route information $Route(U \rightarrow R_i)$ **do**
23: $\mathcal{A} \leftarrow \mathcal{A} - \{$all objects located in $Route(U \rightarrow R_i)\}$
24: **end for**
25: **end while**
26: Return the retrieved route information for all the objects in $\mathcal{A}$

| NDist (km) | $R_1$ | $R_2$ | $R_3$ |
|---|---|---|---|
| $R_1$ | 0 | 12 | 16 |
| $R_2$ | 12 | 0 | 5 |
| $R_3$ | 16 | 5 | 0 |

(a) Between objects.

| NDist (km) | $U_1$ | $U_2$ |
|---|---|---|
| $U_1$ | 0 | 3 |
| $U_2$ | 3 | 0 |

(b) Between users.

Fig. 5. The shortest network distance.

467 seconds which is less than $Time(U_1 \rightarrow R_3) = 550$ seconds. Therefore, $Route(U_1 \rightarrow R_2)$ may not be independent with $Route(U_1 \rightarrow R_3)$ based on Theorem 4.1, i.e., $R_2$ should not be processed in parallel with $R_3$. The next object in $\mathcal{A}_1$ is $R_1$. Since $Time(U_1 \rightarrow R_1) + \frac{NDist(R_1 \rightarrow R_3)}{V_{max}} = 50 + \frac{16{,}000}{30} = 583$ seconds is larger than $Time(U_1 \rightarrow R_3) = 550$ seconds, $R_1$ is added to $\mathcal{A}_p$ and removed from $\mathcal{A}_1$, i.e., $\mathcal{A}_p = \{R_1, R_3\}$ and $\mathcal{A}_1 = \{R_2\}$. The *parallel requesting step* sends external requests for $R_1$ and $R_3$ in parallel to retrieve their route information, i.e., $Route(U_1 \rightarrow R_1)$ and $Route(U_1 \rightarrow R_3)$. Since $R_2$ is located in $Route(U_1 \rightarrow R_3)$ (Figure 4), $R_2$ is removed from $\mathcal{A}_1$ based on the direction sharing optimization. After the *parallel requesting step*, $\mathcal{A}_1$ becomes empty. Therefore, the algorithm returns the retrieved route information from $U_1$ to $R_1$, $R_2$, and $R_3$ to $U_1$ as a query answer.

As a result, although both Algorithm 1 and Algorithm 2 need two external requests for $Q_1$, Algorithm 2 can reduce the query response time by half with the help of the parallel requesting optimization. This is because Algorithm 1 issues the two requests one by one while Algorithm 2 can issue them in parallel.

Similar to $Q_1$, both Algorithm 1 and Algorithm 2 need two external requests for $Q_2$, but Algorithm 2 can reduce the query response time by issuing the two requests in parallel.

## 5 Parallel Requesting Optimization for Concurrent Queries

We have presented the direction sharing and parallel requesting optimizations for a single query. That is the system processes users' queries in a sequential manner. However, such a simple query processing paradigm is not efficient for a system with high workload, e.g., a large number of concurrent queries. In this section, we extend these two optimizations for multiple concurrent $k$-NN queries to further reduce the number of external requests and the query response time.

### 5.1 Theoretical Bases

We first introduce two corollaries based on the *prefix property* and Theorem 4.1, which are theoretical bases for extending the direction sharing and parallel requesting optimizations to process concurrent queries.

**Corollary 5.1.** *Given a route $T = Route(A \rightarrow \ldots \rightarrow X \rightarrow \ldots \rightarrow Z \rightarrow \ldots \rightarrow B)$ with the shortest travel time from location $A$ to location $B$, where $X$ and $Z$ are two intermediate points in $T$, every sub-route in $T$ also has the shortest travel time.*

*Proof.* Suppose that there is a route $T' = Route(X \rightarrow \ldots \rightarrow Y \rightarrow \ldots \rightarrow Z)$, where $Y \notin T$, with a travel time shorter than $Route(X \rightarrow Z)$ in $T$. We could replace $Route(X \rightarrow Z)$ in $T$ with $T'$ to result in a new route with a shorter travel time than $T$. Thus, this assumption contradicts the optimality of $T$. □

Based on Corollary 5.1, the information of a route can be shared with its sub-route; thus, this idea can be used to reduce the number of external requests.

**Corollary 5.2.** *Given the route from location $A$ to location $B$ (i.e., $Route(A \rightarrow B)$), the route from location $C$ to location $D$ in a road network (i.e., $Route(C \rightarrow D)$), and the travel time of $Route(A \rightarrow B)$ is shorter than that of $Route(C \rightarrow D)$ (i.e., $Time(A \rightarrow B) < Time(C \rightarrow D)$), $Route(A \rightarrow B)$ is independent with $Route(C \rightarrow D)$ if the following inequality holds:*

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

where $NDist(C \rightarrow A)$ and $NDist(B \rightarrow D)$ are the shortest network distances of from $C$ to $A$ and from $B$ to $D$, respectively, and $V_{max}$ is the maximum movement speed defined by the user or restricted by the road network.

*Proof.* Similar to the proof of Theorem 4.1, if the inequality is true, $Route(A \rightarrow B)$ is not a sub-route of $Route(C \rightarrow D)$. Since $\frac{NDist(C \rightarrow A)}{V_{\max}}$ and $\frac{NDist(B \rightarrow D)}{V_{\max}}$ are the minimum

possible travel time from $C$ to $A$ and $B$ to $D$, respectively, if $Route(A \rightarrow B)$ is a sub-route of $Route(C \rightarrow D)$, then

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

□

Based on Corollary 5.2, independent routes can be found and parallel requests can be issued for them to retrieve their route information from the directions service. Hence, we can use this idea to reduce the query response time.

### 5.2 Algorithm

Algorithm 3 sketches the pseudo code of the extended direction sharing and parallel requesting optimizations. It inputs include a road network model $G = (V, E)$, a list of outstanding queries $\mathcal{L}$, in which each query $Q_i$ issued by user $U_i$ is along with an answer set $\mathcal{A}_i$, a set of candidate pairs $\mathcal{P}$ of each $Q_i$ and each object $R_j$ in $\mathcal{A}_i$ (i.e., $\langle U_i, R_j \rangle$), and the maximum movement speed $V_{\max}$. For each iteration, Algorithm 3 consists of two iterative steps processing the pairs in $\mathcal{P}$ until $\mathcal{P}$ becomes empty.

(1) **Parallel destination pair set step.** A destination pair set $\mathcal{P}_p$ is initially set to empty to store the pairs for which the system sends external requests in parallel to access their route information (Line 4 in Algorithm 3). The pairs in $\mathcal{P}$ are sorted with their travel time in decreasing order. The pair with the longest travel time is moved to $\mathcal{P}_p$ directly (Line 6). For each pair $\langle U_l, R_m \rangle$ in $\mathcal{P}$, if $\langle U_l, R_m \rangle$ satisfies the inequality of Corollary 5.2 with each pair in $\mathcal{P}_p$, $\langle U_l, R_m \rangle$ can be processed in parallel with other pairs in $\mathcal{P}_p$ because it is independent of them. Hence, $\langle U_l, R_m \rangle$ is inserted into $\mathcal{P}_p$ and is removed from $\mathcal{P}$ (Lines 7 to 19).

(2) **Parallel requesting step.** For all the pairs in $\mathcal{P}_p$, external requests are issued in parallel to the directions service to retrieve their route information. For the retrieved route information of each $\langle U_i, R_j \rangle$ (i.e., $Route(U_i \rightarrow R_j)$), $Route(U_i \rightarrow R_j)$ can be shared with other pairs located in it and in $\mathcal{P}$ by the direction sharing optimization; thus, such pairs are removed from $\mathcal{P}$ (Lines 21 to 24). The algorithm next checks if any query $Q_i$ in the waiting list $\mathcal{L}$ is completed, i.e., all its pairs have been removed from $\mathcal{P}$. For each completed query $Q_i$, the route information from $U_i$ to each object in $\mathcal{A}_i$ is returned to $U_i$, and $Q_i$ is removed from $\mathcal{L}$ (Lines 25 to 30). For a newly received query $Q_j$ with its answer set $\mathcal{A}_i$, $Q_j$ is inserted into $\mathcal{L}$ and all of its pairs $\langle U_i, R_j \rangle$, where $(R_j \in \mathcal{A}_i)$, are inserted into $\mathcal{P}$ (Lines 32 to 33).

**Example.** In the running example, the input of Algorithm 3 includes $\mathcal{L} = \{Q_1, Q_2\}$, $\mathcal{P} = \{\langle U_1, R_1 \rangle, \langle U_1, R_2 \rangle, \langle U_1, R_3 \rangle, \langle U_2, R_1 \rangle, \langle U_2, R_2 \rangle, \langle U_2, R_3 \rangle\}$ (Figure 3), $G = (V, E)$ (Figure 2b), and $V_{max} = 30$ m/s. The shortest network distance between a user and an object is calculated based on $G$ (Figure 5). Algorithm 3 executes two iterations for this example. Table 1 illustrates the computation of the two steps in each iteration in detail.

***In the first iteration.*** The *parallel destination pair set step* sets $\mathcal{P}_p$ to empty and sorts the pairs in $\mathcal{P}$ by their travel time in decreasing order (Row 1a in Table 1). As $\langle U_2, R_3 \rangle$ has the longest travel time, it is moved from $\mathcal{P}$ to $\mathcal{P}_p$(Row 1b). Then, the algorithm checks the independence of each pair in $\mathcal{P}$ with the pairs in $\mathcal{P}_p$ (Rows 1c to 1g):

- For the pairs $\langle U_1, R_3 \rangle$, $\langle U_2, R_2 \rangle$, and $\langle U_1, R_2 \rangle$, since they do not satisfy the inequality of Corollary 5.2 with the pair in $\mathcal{P}_p = \{\langle U_2, R_3 \rangle\}$, they may not be independent with $\langle U_2, R_3 \rangle$, i.e., $\langle U_1, R_3 \rangle$, $\langle U_2, R_2 \rangle$, and $\langle U_1, R_2 \rangle$ should not be processed in parallel with $\langle U_2, R_3 \rangle$; hence, they are not inserted into $\mathcal{P}_p$ (Rows 1c to 1e).
- For the pair $\langle U_2, R_1 \rangle$, it satisfies the inequality of Corollary 5.2 with $\langle U_2, R_3 \rangle$, i.e., $\langle U_2, R_1 \rangle$ can be processed with $\langle U_2, R_3 \rangle$ in parallel; thus, it is moved from $\mathcal{P}$ to $\mathcal{P}_p$ (Row 1f).
- For the last pair $\langle U_1, R_1 \rangle$, since it does not satisfy the inequality of Corollary 5.2 with $\langle U_2, R_3 \rangle$ in $\mathcal{P}_p$, it is not moved to $\mathcal{P}_p$ (Row 1g).

As depicted in Row 1h, the *parallel requesting step* issues two external requests in parallel to the directions service for the two pairs in $\mathcal{P}_p = \{\langle U_2, R_3 \rangle, \langle U_2, R_1 \rangle\}$ to retrieve

**Algorithm 3** Extended direction sharing and parallel requesting optimizations for concurrent queries

```
1: input: G = (V, E), P, L, V_max
2: while P is not empty do
3:   // Parallel destination pair set step
4:   P_p ← {}
5:   Sort the pairs in P with their travel time in decreasing order
6:   Move the pair with the longest travel time from P to P_p
7:   for each pair ⟨U_l, R_m⟩ in P do
8:     independent ← true
9:     for each pair ⟨U_i, R_j⟩ in P_p do
10:      Compute NDist(U_i → U_l) and NDist(R_m → R_j) based on G = (V, E)
11:      if NDist(U_i→U_l)/V_max + Time(U_l → R_m) + NDist(R_m→R_j)/V_max
         ≤ Time(U_i → R_j) then
12:        independent ← false
13:        break
14:      end if
15:    end for
16:    if independent = true then
17:      Insert ⟨U_l, R_m⟩ into P_p and remove it from P
18:    end if
19:  end for
20:  // Parallel requesting step
21:  For all the pairs in P_p, external requests are issued in parallel to the directions services to retrieve their route information
22:  for each retrieved route information Route(U_i → R_j) do
23:    P ← P−{all pairs located in Route(U_i → R_j)}
24:  end for
25:  for each Q_i in L do
26:    if all the pairs of Q_i have been removed from P then
27:      Remove Q_i from L
28:      Return the retrieved route information of all the objects in A_i to its user U_i
29:    end if
30:  end for
31:  for each newly received query Q_j do
32:    L ← L ∪ {Q_j}
33:    P ← P∪ {all pairs for Q_j}
34:  end for
35: end while
```

TABLE 1
Example of the extended direction sharing and parallel requesting optimizations for concurrent queries.

| Iteration - Step | Row | Pair | $\mathcal{P}_p$ | $\mathcal{P}$ | $\mathcal{L}$ | Checking and/or actions |
|---|---|---|---|---|---|---|
| 1st - Parallel destination pair set | 1a | - | $\{\}$ | $\{\langle U_2, R_3\rangle, \langle U_1, R_3\rangle, \langle U_2, R_2\rangle, \langle U_1, R_2\rangle, \langle U_2, R_1\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | The pairs in $\mathcal{P}$ are sorted with their travel time in decreasing order. |
| | 1b | - | $\{\langle U_2, R_3\rangle\}$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_2\rangle, \langle U_1, R_2\rangle, \langle U_2, R_1\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $\langle U_2, R_3\rangle$ is moved from $\mathcal{P}$ to $\mathcal{P}_p$. |
| | 1c | $\langle U_1, R_3\rangle$ | $\{\langle U_2, R_3\rangle\}$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_2\rangle, \langle U_1, R_2\rangle, \langle U_2, R_1\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $\frac{NDist(U_2 \to U_1)}{V_{\max}} + Time(U_1 \to R_3) + \frac{NDist(R_3 \to R_3)}{V_{\max}} = \frac{3,000}{30} + 550 + 0 = 650 < Time(U_2 \to R_3) = 700$ |
| | 1d | $\langle U_2, R_2\rangle$ | $\{\langle U_2, R_3\rangle\}$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_2\rangle, \langle U_1, R_2\rangle, \langle U_2, R_1\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $\frac{NDist(U_2 \to U_2)}{V_{\max}} + Time(U_2 \to R_2) + \frac{NDist(R_2 \to R_3)}{V_{\max}} = 0 + 500 + \frac{5,000}{30} = 667 < Time(U_2 \to R_3) = 700$ |
| | 1e | $\langle U_1, R_2\rangle$ | $\{\langle U_2, R_3\rangle\}$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_2\rangle, \langle U_1, R_2\rangle, \langle U_2, R_1\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $\frac{NDist(U_2 \to U_1)}{V_{\max}} + Time(U_1 \to R_2) + \frac{NDist(R_2 \to R_3)}{V_{\max}} = \frac{3,000}{30} + 300 + \frac{5,000}{30} = 567 < Time(U_2 \to R_3) = 700$ |
| | 1f | $\langle U_2, R_1\rangle$ | $\{\langle U_2, R_3\rangle, \langle U_2, R_1\rangle\}$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_2\rangle, \langle U_1, R_2\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $\frac{NDist(U_2 \to U_2)}{V_{\max}} + Time(U_2 \to R_1) + \frac{NDist(R_1 \to R_3)}{V_{\max}} = 0 + 200 + \frac{16,000}{30} = 733 > Time(U_2 \to R_3) = 700$; thus, $\langle U_2, R_1\rangle$ is moved from $\mathcal{P}$ to $\mathcal{P}_p$. |
| | 1g | $\langle U_1, R_1\rangle$ | $\{\langle U_2, R_3\rangle, \langle U_2, R_1\rangle\}$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_2\rangle, \langle U_1, R_2\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $\frac{NDist(U_2 \to U_1)}{V_{\max}} + Time(U_1 \to R_1) + \frac{NDist(R_1 \to R_3)}{V_{\max}} = \frac{3,000}{30} + 50 + \frac{16,000}{30} = 683 < Time(U_2 \to R_3) = 700$ |
| 1st - Parallel requesting | 1h | - | $\{\}$ | $\{\langle U_1, R_1\rangle\}$ | $\{Q_1\}$ | Two parallel external requests are issued to retrieve $Route(U_2 \to R_3)$ and $Route(U_2 \to R_1)$ that can be shared with $\langle U_1, R_2\rangle$, $\langle U_1, R_3\rangle$, and $\langle U_2, R_2\rangle$. All the pairs of $Q_2$ have been processed. |
| 2nd - Parallel destination pair set | 2a | - | $\{\langle U_1, R_1\rangle\}$ | $\{\}$ | $\{Q_1\}$ | $\langle U_1, R_1\rangle$ is removed from $\mathcal{P}$ to $\mathcal{P}_p$ |
| 2nd - Parallel requesting | 2b | - | $\{\}$ | $\{\}$ | $\{\}$ | One external request is issued to retrieve $Route(U_1 \to R_1)$. All the pairs of $Q_1$ have been processed. |

their route information. Since both $U_1$ and $R_2$ are located in $Route(U_2 \to R_3)$ (Figure 4), $\langle U_1, R_2\rangle$ can share with $Route(U_2 \to R_3)$. Similary, $\langle U_1, R_3\rangle$, and $\langle U_2, R_2\rangle$ can share with $Route(U_2 \to R_3)$ too. Therefore, those pairs (i.e., $\langle U_1, R_2\rangle$, $\langle U_1, R_3\rangle$, and $\langle U_2, R_2\rangle$) are removed from $\mathcal{P}$. After that, all the pairs of $Q_2$ have been processed, so $Q_2$ is removed from the waiting list $\mathcal{L}$ and the route information from the user of $Q_2$ (i.e., $U_2$) to each object in $\mathcal{A}_2$ is returned to $U_2$.

***In the second iteration.*** As $\mathcal{P} = \{\langle U_1, R_1\rangle\}$ is not empty, the algorithm proceeds to execute the second iteration. The *parallel destination pair set step* first moves the only pair $\langle U_1, R_1\rangle$ from $\mathcal{P}$ to $\mathcal{P}_p$ (Row 2a). Then, the *parallel requesting step* issues one external request to the directions service to retrieve the route information of $\langle U_1, R_1\rangle$. All the pairs of $Q_1$ have been processed. The route information from $U_1$ to each object in $Q_1$'s answer set $\mathcal{A}_1$ is returned to $U_1$ (Row 2b). Since $\mathcal{P}$ becomes empty, the algorithm terminates and waits for new queries.

In comparison with Algorithm 1 and Algorithm 2, Algorithm 3 only issues three external requests, while both Algorithm 1 and Algorithm 2 need four external requests. In terms of the query response time, Algorithm 1 issues four external requests sequentially; Algorithm 2 processes $Q_1$ and $Q_2$ one by one by issuing two external requests in parallel for $Q_1$ and two additional requests in parallel for $Q_2$. Algorithm 3 requires two parallel requests in the first iteration and one request in the second iteration. Therefore, Algorithms 2 and 3 have the same query response time, but Algorithm 3 reduces the number of external requests from four to three.

### 5.3 Starvation Problem

In Algorithm 3, the *parallel destination pair set step* simply processes the pairs in $\mathcal{P}$ based on their travel time in decreasing order. The major drawback of this simple processing method is that it could take a long time or never process a pair; a user may suffer from a long query response time or may not be able to get an answer for her/his query. In this section, we address such a starvation problem (Definition 2) by using a performance tuning tool to provide a trade-off between the query response time of individual queries and the number of external requests.

**Definition 2** (Starvation problem)**.** *A pair $\langle U_i, R_j\rangle$ in $\mathcal{P}$ is said to be starved, if its waiting time exceeds a certain time limit (e.g., 5 seconds). The time limit would be set to a default system parameter or a user-specified query parameter.*

*New selection technique.* To address the starvation problem, the pairs in $\mathcal{P}$ are ranked by a score that is determined by their *travel time* and *waiting time*, instead of only ranking them based on their travel time. The score of a pair $\langle U_i, R_j\rangle$ in $\mathcal{P}$ is calculated based on the weighted sum of its normalized *travel time* and *waiting time*, where $0 \leq \alpha \leq 1$:

$$Score(U_i \to R_j) = \alpha \times \frac{WTime(U_i \to R_j)}{MaxWTime} + (1-\alpha) \times \frac{Time(U_i \to R_j)}{MaxTime}, \quad (1)$$

where $WTime(U_i \to R_j)$ is the waiting time of $\langle U_i, R_j\rangle$ since its query $Q_i$ has been received by the system, $Time(U_i \to R_j)$ is the travel time from $U_i$ to $R_j$ computed by our basic $k$-NN query processing (described in Section 3), $MaxWTime$ is the longest waiting time of all the pairs in $\mathcal{P}$ that is used to normalize the waiting time of each pair, $MaxTime$ is the longest travel time of all the pairs in $\mathcal{P}$

**Algorithm 4** The modified parallel destination pair set step of Algorithm 3 with starvation prevention

```
1: Lines 1 to 3 in Algorithm 3
2: P_p ← {}
3: Sort the pairs in P with their scores computed by Equation 1
   in decreasing order
4: Move the pair with the largest score from P to P_p
5: MaxTime_p ← the longest travel time of the pairs in P_p
6: for each pair ⟨U_l, R_m⟩ in P do
7:    if Time(U_l → R_m) > MaxTime_p then
8:       independent ← true
9:    else
10:      Lines 8 to 15 in Algorithm 3
11:   end if
12:   if independent = true then
13:      Insert ⟨U_l, R_m⟩ into P_p and remove it from P
14:      MaxTime_p ← the longest travel time of the pairs in
         P_p
15:   end if
16: end for
17: Lines 20 to 35 in Algorithm 3
```

that is used to normalize the travel time of each pair, and $\alpha$ is a parameter to adjust the importance of the waiting time. A larger value of $\alpha$ leads to a more important role of the waiting time in Equation 1; and thus, a pair with the longer waiting time has a higher priority to be processed. However, it would incur a larger number of external requests issued to the directions service. As a result, it is important to find a good value for $\alpha$ to optimize the system performance and avoid the starvation problem. We provide empirical studies to study this problem in Section 6.

**Algorithm.** We present two modifications to incorporate the new selection technique into Algorithm 3. (1) The pairs in $\mathcal{P}$ are sorted by their scores computed by Equation 1 in decreasing order, and the pair with the largest score is moved from $\mathcal{P}$ to $\mathcal{P}_p$ (Lines 3 to 4 in Algorithm 4). A situation will happen that a pair with the larger travel time may be selected into $\mathcal{P}_p$ later than the pair with the smaller travel time, which is impossible to happen in Algorithm 3 as it processes pairs in $\mathcal{P}$ only based on the decreasing order of their travel time; hence, a new variable called $MaxTime_p$ keeps track of the longest travel time of the pairs in $\mathcal{P}_p$ (Line 5). (2) We have a new condition to select a pair to be processed in parallel. According to Corollary 5.2, if a pair $\langle U_l, R_m\rangle$ in $\mathcal{P}$ has a longer travel time than $MaxTime_p$ (i.e., the travel time of $\langle U_l, R_m\rangle$ is larger than that of any pair in $\mathcal{P}_p$), the route information of the pairs in $\mathcal{P}_p$ cannot be shared with $\langle U_l, R_m\rangle$ based on the direction sharing optimization, so $\langle U_l, R_m\rangle$ is independent with any pair in $\mathcal{P}_p$; and thus, it is moved to from $\mathcal{P}$ to $\mathcal{P}_p$ (Lines 7 to 8 in Algorithm 4). The pairs in $\mathcal{P}_p$ will be processed in parallel (Line 17 in Algorithm 4).

**Example.** Suppose that the waiting time of $Q_1$ and $Q_2$ are 100 ms and 50 ms, respectively. Figure 6 shows the scores of the pairs in the candidate pair set $\mathcal{P}$ based on three different $\alpha$ values.

Table 2 depicts an example for Algorithm 4, where $\alpha = 0.5$. In the first iteration, the *parallel destination pair set step* first sorts the pairs in $\mathcal{P}$ based on their scores in decreasing order (Row 1a). Then, it moves the pair with the largest score (i.e., $\langle U_1, R_3\rangle$) from $\mathcal{P}$ to $\mathcal{P}_p$ and sets

| $\mathcal{P}$ | ***WTime*** (ms) *MaxWTime*=100 | ***Time*** (Seconds) *MaxTime*=700 | ***Score***$(=\alpha\times\frac{WTime}{MaxWTime}+(1-\alpha)\times\frac{Time}{MaxTime})$ | | |
|---|---|---|---|---|---|
| | | | α=0 | α=0.5 | α=1 |
| $\langle U_1, R_1\rangle$ | 100 | 50 | 0.07 | 0.54 | 1 |
| $\langle U_1, R_2\rangle$ | 100 | 300 | 0.43 | 0.72 | 1 |
| $\langle U_1, R_3\rangle$ | 100 | 550 | 0.79 | 0.9 | 1 |
| $\langle U_2, R_1\rangle$ | 50 | 200 | 0.29 | 0.40 | 0.5 |
| $\langle U_2, R_2\rangle$ | 50 | 500 | 0.71 | 0.61 | 0.5 |
| $\langle U_2, R_3\rangle$ | 50 | 700 | 1 | 0.75 | 0.5 |

Fig. 6. The score of each pair based on three $\alpha$ values.

$MaxTime_p = Time(U_1 \rightarrow R_3) = 550$ (Row 1b). Since the travel time of the next pair $\langle U_2, R_3\rangle$ is larger than $MaxTime_p$, so $\langle U_2, R_3\rangle$ is moved to $\mathcal{P}_p$ and $MaxTime_p$ is updated to 700 based on our new selection technique (i.e., Lines 7 to 8 of Algorithm 4) (Row 1c). For the next three pairs, $\langle U_1, R_2\rangle$, $\langle U_2, R_2\rangle$, and $\langle U_1, R_1\rangle$, their travel times are less than $MaxTime_p$, and they may not be independent with the two pairs in $\mathcal{P}_p$ since they do not satisfy the inequality of Corollary 5.2, they remain in $\mathcal{P}$ (Rows 1d to 1f). For the last pair $\langle U_2, R_1\rangle$, although its travel time is smaller than $MaxTime_p$, it is independent with the two pairs in $\mathcal{P}_p$ based on Corollary 5.2; thus, it is moved to $\mathcal{P}_p$. The *parallel requesting step* issues an external request for each pair in $\mathcal{P}_p = \{\langle U_1, R_3\rangle, \langle U_2, R_3\rangle, \langle U_2, R_1\rangle\}$ (Row 1h). The retrieved route information can be used to find $Route(U_1 \rightarrow R_2)$ and $Route(U_2 \rightarrow R_2)$, so $\langle U_1, R_2\rangle$ and $\langle U_2, R_2\rangle$ are removed from $\mathcal{P}$. Since all the pairs of $Q_2$ have been processed, the required route information is returned to the user of $Q_2$.

In the second iteration, there is only one pair $\langle U_1, R_1\rangle$ in $\mathcal{P}$; hence, the *parallel destination pair set step* moves $\langle U_1, R_1\rangle$ from $\mathcal{P}$ to $\mathcal{P}_p$ and sets $MaxTime_p$ to 50 (Row 2a). Since $\mathcal{P}$ is empty, the algorithm proceeds to the *parallel requesting step* that issues one external request to retrieve $Route(U_1 \rightarrow R_1)$ (Row 2b). After that, all the pairs of $Q_1$ have been processed, their route information is sent to the user of $Q_1$.

In comparison to Table 1, Algorithm 4 requires one more external request than Algorithm 3 (i.e., four requests), and keeps the same number of iterations. However, the pair $\langle U_1, R_3\rangle$ is selected into the destination pair set $\mathcal{P}_p$ in the first iteration by Algorithm 4, but it is not selected by Algorithm 3.

## 6 Experimental Evaluation

In this section, we first give our evaluation model in Section 6.1, and then evaluate the performance of the parallel requesting optimization for a single query (Section 6.2), and the performance of the parallel requesting optimization extended for multiple concurrent queries (Section 6.3).

### 6.1 Evaluation Model

**Evaluated approaches.** For each evaluated approach, our basic $k$-NN query processing (described in Section 3) is the first one to be executed to find a $k$-NN query answer for each user query through the matrix service. Then, one of the following algorithms is executed to access the detailed route information from each querying user to each of the $k$ objects in her/his query answer through the directions service:

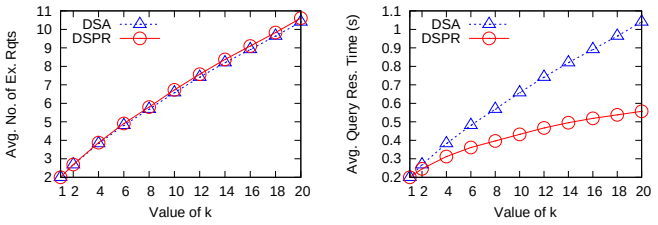
TABLE 2
Example of the extended optimizations for concurrent queries with the starvation prevention ($\alpha = 0.5$).

| Iteration - Step | Row | Pair | $\mathcal{P}_p$ | $\mathcal{P}$ | $\mathcal{L}$ | Checking and/or actions |
|---|---|---|---|---|---|---|
| 1st - Parallel destination pair set | 1a | - | $\{\}$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_3\rangle, \langle U_1, R_2\rangle, \langle U_2, R_2\rangle, \langle U_1, R_1\rangle, \langle U_2, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | The pairs in $\mathcal{P}$ are sorted with their scores calculated by $Score(U_i \rightarrow R_j) = 0.5 \times \frac{WTime(U_i \rightarrow R_j)}{MaxWTime} + 0.5 \times \frac{Time(U_i \rightarrow R_j)}{MaxTime}$ in decreasing order. |
| | 1b | - | $\{\langle U_1, R_3\rangle\}$; $MaxTime_p = 550$ | $\{\langle U_2, R_3\rangle, \langle U_1, R_2\rangle, \langle U_2, R_2\rangle, \langle U_1, R_1\rangle, \langle U_2, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $\langle U_1, R_3\rangle$ is moved from $\mathcal{P}$ to $\mathcal{P}_p$. |
| | 1c | $\langle U_2, R_3\rangle$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_3\rangle\}$; $MaxTime_p = 700$ | $\{\langle U_1, R_2\rangle, \langle U_2, R_2\rangle, \langle U_1, R_1\rangle, \langle U_2, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $Time(U_2 \rightarrow R_3) = 700 > MaxTime_p = 550$; thus, $\langle U_2, R_3\rangle$ is moved from $\mathcal{P}$ to $\mathcal{P}_p$. |
| | 1d | $\langle U_1, R_2\rangle$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_3\rangle\}$; $MaxTime_p = 700$ | $\{\langle U_1, R_2\rangle, \langle U_2, R_2\rangle, \langle U_1, R_1\rangle, \langle U_2, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $Time(U_1 \rightarrow R_2) = 300 < MaxTime_p = 700$; and $\frac{NDist(U_1 \rightarrow U_1)}{V_{\max}} + Time(U_1 \rightarrow R_2) + \frac{NDist(R_2 \rightarrow R_3)}{V_{\max}} = 0 + 300 + \frac{5,000}{30} = 467 < Time(U_1 \rightarrow R_3) = 550$ |
| | 1e | $\langle U_2, R_2\rangle$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_3\rangle\}$; $MaxTime_p = 700$ | $\{\langle U_1, R_2\rangle, \langle U_2, R_2\rangle, \langle U_1, R_1\rangle, \langle U_2, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $Time(U_2 \rightarrow R_2) = 500 < MaxTime_p = 700$; and $\frac{NDist(U_1 \rightarrow U_2)}{V_{\max}} + Time(U_2 \rightarrow R_2) + \frac{NDist(R_2 \rightarrow R_3)}{V_{\max}} = \frac{3,000}{30} + 500 + \frac{5,000}{30} = 767 > Time(U_1 \rightarrow R_3) = 550$, but $\frac{NDist(U_2 \rightarrow U_2)}{V_{\max}} + Time(U_2 \rightarrow R_2) + \frac{NDist(R_2 \rightarrow R_3)}{V_{\max}} = 0 + 500 + \frac{5,000}{30} = 667 < Time(U_2 \rightarrow R_3) = 700$ |
| | 1f | $\langle U_1, R_1\rangle$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_3\rangle\}$; $MaxTime_p = 700$ | $\{\langle U_1, R_2\rangle, \langle U_2, R_2\rangle, \langle U_1, R_1\rangle, \langle U_2, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $Time(U_1 \rightarrow R_1) = 50 < MaxTime_p = 700$; and $\frac{NDist(U_1 \rightarrow U_1)}{V_{\max}} + Time(U_1 \rightarrow R_1) + \frac{NDist(R_1 \rightarrow R_3)}{V_{\max}} = 0 + 50 + \frac{16,000}{30} = 583 > Time(U_1 \rightarrow R_3) = 550$, but $\frac{NDist(U_2 \rightarrow U_1)}{V_{\max}} + Time(U_1 \rightarrow R_1) + \frac{NDist(R_1 \rightarrow R_3)}{V_{\max}} = \frac{3,000}{30} + 50 + \frac{16,000}{30} = 683 < Time(U_2 \rightarrow R_3) = 700$ |
| | 1g | $\langle U_2, R_1\rangle$ | $\{\langle U_1, R_3\rangle, \langle U_2, R_3\rangle, \langle U_2, R_1\rangle\}$; $MaxTime_p = 700$ | $\{\langle U_1, R_2\rangle, \langle U_2, R_2\rangle, \langle U_1, R_1\rangle\}$ | $\{Q_1, Q_2\}$ | $Time(U_2 \rightarrow R_1) = 200 < MaxTime_p = 700$, but $\frac{NDist(U_1 \rightarrow U_2)}{V_{\max}} + Time(U_2 \rightarrow R_1) + \frac{NDist(R_1 \rightarrow R_3)}{V_{\max}} = \frac{3,000}{30} + 200 + \frac{16,000}{30} = 833 > Time(U_1 \rightarrow R_3) = 550$ and $\frac{NDist(U_2 \rightarrow U_2)}{V_{\max}} + Time(U_2 \rightarrow R_1) + \frac{NDist(R_1 \rightarrow R_3)}{V_{\max}} = 0 + 200 + \frac{16,000}{30} = 733 > Time(U_2 \rightarrow R_3) = 700$; thus, $\langle U_2, R_1\rangle$ is moved from $\mathcal{P}$ to $\mathcal{P}_p$. |
| 1st - Parallel requesting | 1h | - | $\{\}$ | $\{\langle U_1, R_1\rangle\}$ | $\{Q_1\}$ | Three parallel external requests are issued to retrieve $Route(U_1 \rightarrow R_3)$, $Route(U_2 \rightarrow R_3)$ and $Route(U_2 \rightarrow R_1)$ that can be shared with $\langle U_1, R_2\rangle$ and $\langle U_2, R_2\rangle$. All the pairs of $Q_2$ have been processed. |
| 2nd - Parallel destination pair set | 2a | - | $\{\langle U_1, R_1\rangle\}$; $MaxTime_p = 50$ | $\{\}$ | $\{Q_1\}$ | $\langle U_1, R_1\rangle$ is moved from $\mathcal{P}$ to $\mathcal{P}_p$. |
| 2nd - Parallel requesting | 2b | - | $\{\}$ | $\{\}$ | $\{\}$ | One external request is issued to retrieve $Route(U_1 \rightarrow R_1)$. All the pairs of $Q_1$ have been processed. |

1) **Direction Sharing Algorithm** (DSA). To our best knowledge, our previous work [22], [23] is the state-of-the-art $k$-NN query processing algorithm using spatial mashups in time-dependent road networks. We use DSA as a baseline algorithm in the experiments (Algorithm 1).
2) **Direction Sharing and Parallel Requesting algorithm for single query** (DSPR). This algorithm combines the direction sharing and parallel requesting optimizations (Algorithm 2).
3) **Direction Sharing and Parallel Requesting algorithm for Multiple concurrent queries** (DSPR-M). This algorithm extends the direction sharing and parallel requesting optimizations for multiple concurrent queries (Algorithm 3).
4) **Direction Sharing and Parallel Requesting algorithm with Starvation prevention for Multiple concurrent queries** (DSPR-SM). This algorithm modifies the *parallel destination pair set step* in DSPR-M to prevent the starvation problem (Algorithm 4).

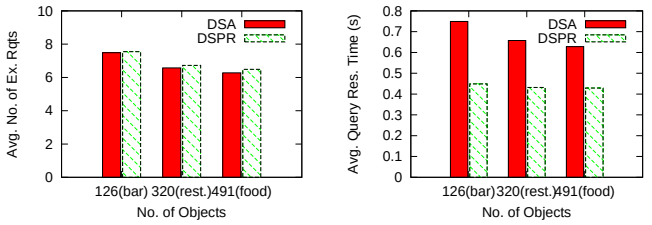
**Performance metrics.** We evaluate the performance of the four evaluated approaches in terms of two metrics: (1) the average number of external web mapping requests per user query (including both the matrix service and the directions service); and (2) the average response time per user query. The response time of a query is the time from the time when the query is received by the system to the time when the answer is returned to the querying user; thus, it includes the local CPU processing time, the communication time between the system and the web mapping service, and the remote processing time at MapQuest Maps [39]. Based on our experiment, the average response time per request from MapQuest Maps is about 98.6 ms.

**Experiment settings.** We implemented the four evaluated algorithms using C++ and MapQuest Maps [39] with a real road network of Hennepin County, MN, USA [40]. We selected an area of $8 \times 8$ 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. Three real data sets within the selected area were collected from Google Places API [41], i.e., 126 bars, 320 restaurants, and 491 food places. Unless mentioned otherwise, we use the real restaurant data set and uniformly generate 50,000 queries at a rate of 500 queries per second (i.e., the experiment duration is 100 seconds). The default requested number of nearest objects of $k$-NN queries (i.e., the value of $k$) is 10, and the maximum movement speed is 120 km per hour. Since most of web mapping service providers can support a very large number of parallel requests, we consider no limit on the maximum number of parallel requests that can be processed by MapQuest Maps.



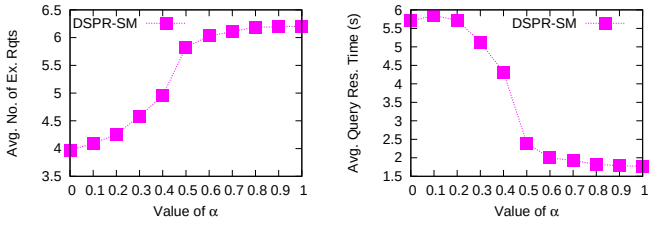

(a) No. of external requests (b) Query response time

Fig. 7. Effect of the value of $k$.




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

Fig. 8. Effect of the number of objects.




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

Fig. 9. Effect of the value of $\alpha$.

### 6.2 The Performance of the Parallel Requesting Optimization

In this section, we evaluate the scalability and efficiency of the parallel requesting optimization by comparing the performance of the direction sharing optimization (DSA) and the algorithm with the direction sharing and parallel requesting optimizations (DSPR).

#### 6.2.1 *Effect of the Requested Number of Nearest Objects ($k$)*

Figure 7 shows the performance of DSA and DSPR with respect to various required numbers of nearest objects (i.e., the value of $k$) from 1 to 20. Without the direction sharing optimization, each object in a query answer requires one external request to retrieve the route information from the querying user to the object, i.e., the server issues $k$ requests for each query. Figure 7a shows that both the DSA and DSPR can effectively reduce the number of external requests. For example, when $k = 20$, the average number of external requests of both DSA and DSPR is less than 11. When $k$ gets larger, a query answer contains more objects. Thus, the server needs to issue more external requests to MapQuest Maps, which results in a longer query response time.

The number of external requests of DSPR is slightly larger than that of DSA (Figure 7a). The main reason is that DSA issues external requests to the web mapping service in a sequential manner (i.e., one request by one request), while DSPR issues external requests in a parallel manner (i.e., a certain number of external requests are issued at the same time). As a result, DSA has a higher chance to share the route information of an external request with more outstanding requests than DSPR, because the route information of an external request cannot be shared with other parallel requests. It is important to note that the performance of DSPR is comparable to DSA in terms of the number of external requests (Figure 7a), but DSPR is much more efficient than DSA in terms of the query response time (Figure 7b). Thus, the experimental results show the effectiveness of DSPR.

#### 6.2.2 *Effect of the Number of Objects*

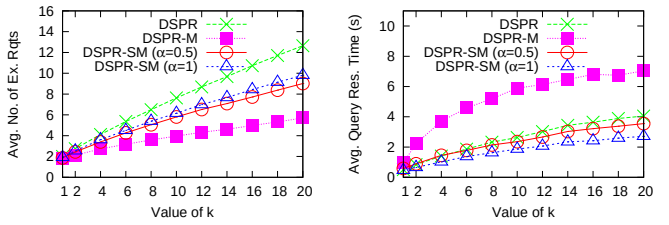
Figure 8 shows the performance of DSA and DSPR with respect to the three real data sets, i.e., bars, restaurants, and food places. Similar to the previous experiment, DSPR significantly reduces the query response time and requires a little bit more external requests than DSA, as depicted in Figures 8a and 8b, respectively. It is interesting to see that both the average number of external mapping requests (Figure 8a) and the average query response time (Figure 8b) of DSA and DSPR drop, as the number of objects gets larger. This is because a larger data set results in higher object density in the road network; hence, the route information from a querying user to an object has a higher chance to share with other objects. Therefore, DSPR is more scalable than DSA for a large number of objects.

### 6.3 The Performance of the Parallel Requesting Optimization for Multiple Concurrent Queries

In this section, we focus on the algorithms designed for multiple concurrent queries DSPR-M and DSPR-SM and consider DSPR as the baseline algorithm to compare their performance. To evaluate the tuning parameter $\alpha$ of DSPR-SM, we vary $\alpha$ from zero to one in Section 6.3.1. In addition, we consider three $\alpha$ values ($\alpha = 0$, $\alpha = 0.5$, and $\alpha = 1$) for all the experiments in this section. Note that when $\alpha = 0$, DSPR-SM is exactly the same as DSPR-M; when $\alpha = 1$, DSPR-SM processes pairs in $\mathcal{P}$ based on their waiting time. Since DSPR performs much better than DSA in terms of the query response time and it incurs a slightly more external requests than DSA, as depicted in Section 6.2, the performance of DSA is not shown in this section.

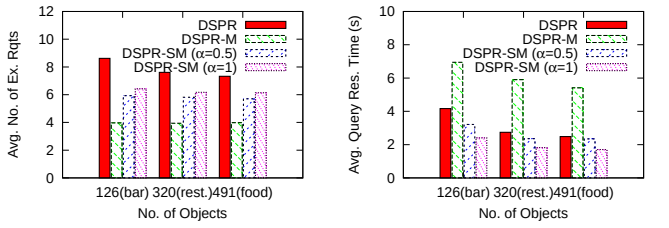
#### 6.3.1 *Effect of Tuning Parameter $\alpha$*

Figure 9 shows the performance of DSPR-SM with various $\alpha$ values. When $\alpha$ increases from zero to one, the average number of external requests increases from 3.97 to 6.21 (56.4% increase as shown in Figure 9a), while the average query response time decreases from 5.7 seconds to 1.77 seconds (68.9% decrease as shown in Figure 9b). The results unveil the importance of $\alpha$ to the performance of DSPR-SM, which is consistent with our theoretical analysis in Section 5.3, i.e., the larger $\alpha$ value leads to a shorter query response time but incurs a larger number of external requests. Most of web mapping service providers will



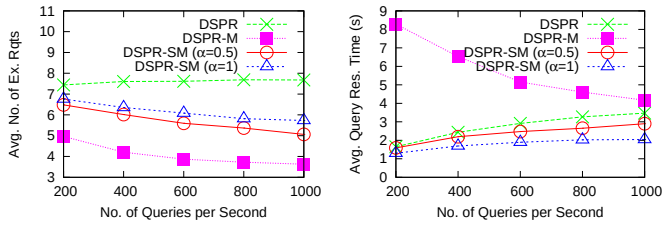

(a) No. of external requests (b) Query response time

Fig. 10. Effect of the value of $k$.



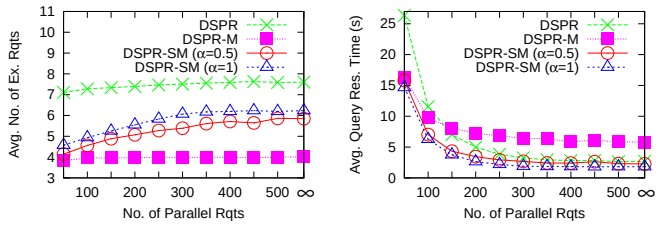

(a) No. of external requests (b) Query response time

Fig. 11. Effect of the number of objects.




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

Fig. 12. Effect of the number of queries per second.




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

Fig. 13. Effect of the number of parallel requests.

charge for a system if its number of requests exceeds a certain threshold [9]–[11]. Therefore, an LBS provider may not want to minimize the query response time for all users. Instead, it could provide different service quality (i.e., query response time) for different levels of users, e.g., a larger $\alpha$ for commercial users, while a smaller $\alpha$ for evaluation users.

#### 6.3.2 *Effect of the Requested Number of Nearest Objects ($k$)*

Figure 10 depicts the performance of DSPR, DSPR-M and DSPR-SM ($\alpha = 0.5$ and 1) with respect to various requested numbers of nearest objects (i.e., $k$) from 1 to 20. As analyzed in Section 6.2.1, a larger $k$ value leads to a larger number of external requests and higher query response time. Although DSPR-M always records the smallest number of external requests (Figure 10a), its query response time is much higher than other algorithms (Figure 10b). This is because some queries in DSPR-M suffer from a very long query response time, as discussed in Section 5.3. These results prove our motivation behind DSPR-SM. Both DSPR-SM ($\alpha = 0.5$) and DSPR-SM ($\alpha = 1$) perform better than DSPR in terms of both the number of external requests and the query response time. When $\alpha = 1$, DSPR-SM achieves the best query response time compared with $\alpha = 0$ (i.e., DSPR-M) and $\alpha = 0.5$, but it needs more external requests. The results of this experiment indicate that $\alpha$ is an effective tuning parameter to balance between the number of external requests and the query response time.

#### 6.3.3 *Effect of the Number of Objects*

Figure 11 gives the results of DSPR, DSPR-M and DSPR-SM ($\alpha = 0.5$ and 1) with respect to the three real data sets. Similar to Section 6.2.2, the average number of external requests and the query response time of all the algorithms decrease with the increase of the data set size, as depicted in Figures 11a and b, respectively. The reason is that a higher object density results in a higher chance for the direction sharing optimization to share route information among queries. Thus, DSPR-SM not only guarantees the query response time of queries, but it is also scalable for a large number of objects.

#### 6.3.4 *Effect of Query Arrival Rates*

Figure 12 gives the results of the algorithms with respect to various numbers of queries per second (i.e., query arrival rate) from 200 to 1,000. As shown in Figure 12a, the average number of external requests of DSPR is only slightly affected by the query arrival rate because it processes queries sequentially. However, the number of external request of DSPR-M and DSPR-SM decreases, as the query arrival rate gets higher. This is because the more concurrent queries, the higher chance for the direction sharing optimization to share the route information retrieved from MapQuest Maps among the queries. Our proposed algorithms are scalable for a large number of concurrent queries. It is expected that the query response time increases as the query arrival rate gets higher. Figure 12b shows that DSPR-SM is also scalable for a high query arrival rate, because the increase rate of its query response time is much lower than that of the query arrival rate. For all query arrival rates, DSPR-M performs worse than DSPR-SM. Especially, when the query arrival rate is small, DSPR-M results in a long query response time because some queries suffer from a long query response time. When there are more concurrent queries, there is a higher chance for queries to share the route information retrieved from MapQuest Maps; hence, the query response time of DSPR-M becomes better when the query arrival rate increases.

#### 6.3.5 *Effect of the Number of Parallel Requests*

Figure 13 depicts the performance of DSPR, DSPR-M and DSPR-SM with the increase of maximum numbers of paral-

lel requests (limited by the web mapping service provider) from 50 to infinity. The number of external requests of DSPR and DSPR-M is only slightly affected by the number of parallel requests (Figure 13a) because they tend to issue external requests for retrieving independent routes. On the other hand, the number of external requests of DSPR-SM increases when the number of parallel requests gets higher. This is because DSPR-SM issues more parallel requests for dependent routes that weaken the sharing power of the direction sharing optimization. It is expected that when the system is able to send more parallel requests, it achieves a better query response time (Figure 13b). Given the default value of $k$ and query arrival rate, the maximum number of parallel requests required by all the algorithms is about 500; and therefore, the maximum number of parallel requests has almost no effect on the query response time after it is larger than 500.

## 7 Conclusion

In this paper, we present the server-side Spatial Mashup Service (SMS) that utilizes the real-time direction information retrieved from web mapping services to process $k$-nearest-neighbor ($k$-NN) queries based on the travel time in road networks. In SMS, we first describe how to extend the direction sharing optimization to use both the matrix and direction services, and then propose a parallel requesting optimization to process external requests in parallel for a single query to improve the system performance (i.e., the number of external requests and the query response time). To further improve the system scalability, the parallel requesting optimization is extended for multiple concurrent queries with the consideration of the starvation problem. To evaluate the performance of our SMS, we have conducted extensive experiments using MapQuest Maps, a real road network, three real object data sets, and a synthetic user data set. The experimental results show that our SMS significantly improves the system performance, and is scalable for large numbers of users and objects.

## Acknowledgments

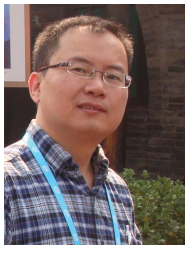
This work was supported in part by the Fundamental Research Funds for the Central Universities in China (Project No. JUSRP11557), the National Natural Science Foundation of China (Project No. 61379130 and 61379038), City University of Hong Kong through an applied research grant (Project No. 9667095) and a strategic research grant (Project No. 7004218).

## References

[1] A. Vancea, M. Grossniklaus, and M. C. Norrie, "Database-driven web mashups," in *ICWE*, 2008.
[2] HousingMaps, "http://www.housingmaps.com."
[3] Yahoo! Pipes, "http://pipes.yahoo.com/pipes."
[4] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding mashup development," *IEEE Internet Computing*, vol. 12, no. 5, pp. 44–52, 2008.
[5] S. Roy Chowdhury, C. Rodríguez, F. Daniel, and F. Casati, "Baya: Assisted mashup development as a service," in *WWW*, 2012.
[6] S. Endrullis, A. Thor, and E. Rahm, "Entity search strategies for mashup applications," in *ICDE*, 2012.
[7] K. Huang, Y. Fan, and W. Tan, "An empirical study of programmable web: A network analysis on a service-mashup system," in *ICWS*, 2012.
[8] ProgrammableWeb, "http://www.programmableweb.com."
[9] Google Maps, "http://maps.google.com."
[10] MapQuest Maps, "http://www.mapquestapi.com."
[11] Microsoft Bing Maps, "http://www.bing.com/maps."
[12] Yahoo! Maps, "http://maps.yahoo.com."
[13] Baidu Maps, "http://map.baidu.com/."
[14] C. S. Jensen, "Database aspects of location-based services," in *Location-Based Services*. Morgan Kaufmann, 2004, pp. 115–148.
[15] D. L. Lee, M. Zhu, and H. Hu, "When location-based services meet databases," *Mobile Information Systems*, vol. 1, no. 2, pp. 81–90, 2005.
[16] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao, "Query processing in spatial network databases," in *VLDB*, 2003.
[17] L. H. U, H. J. Zhao, M. L. Yiu, Y. Li, and Z. Gong, "Towards online shortest paths computation," *IEEE TKDE*, vol. PP, no. 99, pp. 1–1, 2013.
[18] B. Yang, C. Guo, and C. S. Jensen, "Travel cost inference from sparse, spatio temporally correlated time series using Markov models," *PVLDB*, vol. 6, no. 9, pp. 769–780, 2013.
[19] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: driving directions based on taxi trajectories," in *GIS*, 2010.
[20] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi, "Efficient $k$-nearest neighbor search in time-dependent spatial networks," in *DEXA*, 2010.
[21] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa, "Preference query evaluation over expensive attributes," in *CIKM*, 2010.
[22] D. Zhang, C.-Y. Chow, Q. Li, X. Zhang, and Y. Xu, "Efficient evaluation of k-NN queries using spatial mashups," in *SSTD*, 2011.
[23] ——, "SMashQ: Spatial mashup framework for k-NN queries in time-dependent road networks," *Distributed and Parallel Databases*, vol. 31, no. 2, pp. 259–287, 2013.
[24] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *SIGMOD*, 2008.
[25] X. Huang, C. S. Jensen, and S. Saltenis, "The islands approach to nearest neighbor querying in spatial networks," in *SSTD*, 2005.
[26] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi, "Towards $k$-nearest neighbor search in time-dependent spatial network databases," in *DNIS*, 2010.
[27] B. George, S. Kim, and S. Shekhar, "Spatio-temporal network databases and routing algorithms: A summary of results," in *SSTD*, 2007.
[28] B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in *EDBT*, 2008.
[29] N. Bruno, L. Gravano, and A. Marian, "Evaluating top-$k$ queries over web-accessible databases," in *ICDE*, 2002.
[30] K. C.-C. Chang and S.-W. Hwang, "Minimal probing: Supporting expensive predicates for top-$k$ queries," in *SIGMOD*, 2002.
[31] D. Zhang, C.-Y. Chow, Q. Li, X. Zhang, and Y. Xu, "Using parallel spatial mashup model to process k-nn queries," in *ER Workshops*, 2013.
[32] L. Alarabi, A. Eldawy, R. Alghamdi, and M. F. Mokbel, "Tareeg: A mapreduce-based system for extracting spatial data from openstreetmap," in *GIS*, 2014.
[33] J. Ballesteros, A. Cary, and N. Rishe, "SpSJoin: Parallel spatial similarity joins," in *GIS*, 2011.
[34] X. Zhou, D. J. Abel, and D. Truffet, "Data partitioning for parallel spatial join processing," *GeoInformatica*, vol. 2, pp. 175–204, 1998.
[35] G. Luo, J. Naughton, and C. Ellmann, "A non-blocking parallel spatial join algorithm," in *ICDE*, 2002.
[36] The Google Distance Matrix API, "https://developers.google.com/maps/documentation/distancematrix."
[37] MapQuest Directions Web Service - Route Matrix, "http://www.mapquestapi.com/directions/\#matrix."
[38] The Google Directions API, "https://developers.google.com/maps/documentation/directions."
[39] MapQuest Directions Web Service, "http://www.mapquestapi.com/directions."
[40] TIGER/Line Shapefiles 2009 for: Hennepin County, Minnesota, "http://www2.census.gov/cgi-bin/shapefiles2009/county-files?county=27053."
[41] The Google Places API, "https://developers.google.com/places/."

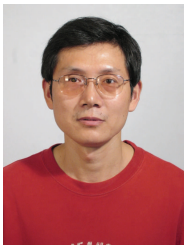
**Detian Zhang** received the joint PhD degree in computer science from the University of Science and Technology of China and City University of Hong Kong in 2014. He is currently a lecture with the School of Digital Media, Jiangnan University, Wuxi, China. His research interests include location-based services, spatio-temporal databases, and wireless networks.

**Chi-Yin Chow** recevied B.A. and MPhil degrees in computing science from The Hong Kong Polytechnic University in 2002 and 2005, respectively, and the M.S. and Ph.D. degrees from the University of Minnesota-Twin Cities in 2008 and 2010, respectively. He is currently an assistant professor in Department of Computer Science, City University of Hong Kong. His research interests include data analytics, spatial and spatio-temporal databases, GIS, data privacy, and mobile computing. He is the co-founder and co-organizer of ACM SIGSPATIAL MobiGIS 2012, 2013, and 2014.



**Qing Li** (SM07) received the B.Eng. degree from Hunan University, Changsha, China, and the M.Sc. and Ph.D. degrees from the University of Southern California, Los Angeles, all in computer science. He is currently a Professor with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. His current research interests include dynamic object modeling, multimedia and mobile information retrieval and management, distributed databases and data warehousing/mining, and workflow management and web services.

**Xinming Zhang** received the BE and ME degrees in electrical engineering from China University of Mining and Technology, Xuzhou, China, in 1985 and 1988, respectively, and the PhD degree in computer science and technology from the University of Science and Technology of China, Hefei, China, in 2001. Since 2002, he has been with the faculty of the University of Science and Technology of China, where he is currently a Professor with the School of Computer Science and Technology. From September 2005 to August 2006, he was a visiting Professor with the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea. His research interest includes wireless networks. He has published more than 70 papers in wireless ad hoc and sensor networks. He is a member of IEEE.



**Yinlong Xu** received his B.S. in Mathematics from Peking University in 1983, and MS and Ph.D in Computer Science from University of Science & Technology of China (USTC) in 1989 and 2004 respectively. He is currently a professor with the School of Computer Science & Technology at USTC. Prior to that, he served the Department of Computer Science & Technology at USTC as an assistant professor, a lecturer, and an associate professor. Currently, he is leading a group of research students working on storage system, high performance computing and network economics. His research interests include fault-tolerant storage system, non-volatile memory, I/O efficient algorithm, network economics, etc. He received the Excellent Ph.D Advisor Award of Chinese Academy of Sciences in 2006 and Baosteel Excellent Teacher Award in 2014.