Real-Time Route Search by Locations

Lisi Chen,[1,2] **Shuo Shang,**[1*] **Tao Guo**[3]
[1]UESTC, China
[2]Inception Institute of Artificial Intelligence, UAE
[3]Google, Singapore
{chenlisi.cs, jedi.shang}@gmail.com, darkgt@google.com

Abstract

With the proliferation of GPS-based data (e.g., routes and trajectories), it is of great importance to enable the functionality of real-time route search and recommendations. We define and study a novel Continuous Route-Search-by-Location (C-RSL) problem to enable real-time route search by locations for a large number of users over route data streams. Given a set of C-RSL queries where each query q contains a set of places $q.O$ to visit and a threshold $q.\theta$, we continuously feed each query q with routes that has similarity to $q.O$ no less than $q.\theta$. We also extend our proposal to support top-k C-RSL problem where each query continuously maintains k most similar routes. The C-RSL problem targets a variety of applications, including real-time route planning, ridesharing, and other location-based services that have real-time demand. To enable efficient route matching on a large number of C-RSL queries, we develop novel parallel route matching algorithms with good time complexity. Extensive experiments with real data offer insight into the performance of our algorithms, indicating that our proposal is capable of achieving high efficiency and scalability.

Introduction

With the continued proliferations of GPS-enabled devices (e.g., vehicle navigation systems and smart phones), online map-based services (e.g., Google Maps), and ridesharing services (e.g., DiDi, Uber, and Grab), travel route data is being generated rapidly. For example, the average number of new taxi trips per day from New York City in 2017 is well over 300K[1]. The availability of massive-scale route data fosters a line of research on RSL-query: Given a collection of routes and a set of query locations, finds a subset of routes that are spatially close to the query locations (Chen et al. 2010; Shang et al. 2012; 2014).

In most existing studies, the RSL query is defined as a one-time query that search from a static collection of routes. However, such one-time query is not capable of delivering users instant results or keeping them updated with most recent results over a stream of route data (Li et al. 2013; Chen, Cong, and Cao 2013). This motivates us to study a novel

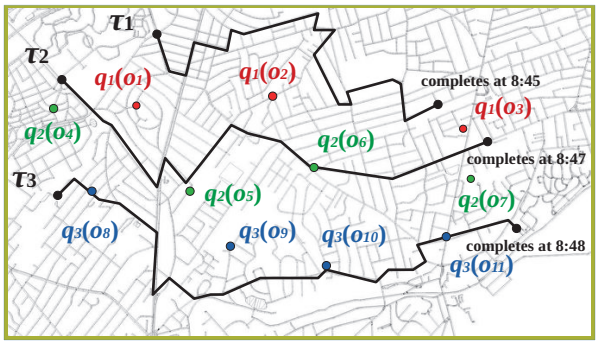


Figure 1: An example of the C-RSL problem

Continuous Route-Search-by-Location (C-RSL) problem. Consider the example in Figure 1, where $q_1 = \{o_1, o_2, o_3\}$, $q_2 = \{o_4, o_5, o_6, o_7\}$, and $q_3 = \{o_8, o_9, o_{10}, o_{11}\}$ are three C-RSL queries. Here, o_1–o_{11} are query locations (user-specified places), and τ_1, τ_2, and τ_3 are routes that arrive in a streaming fashion. When the trip of τ_1 completes (8:45), we compute the similarity between τ_1 and each C-RSL query (i.e., Sim$(\cdot,\cdot)$). If the similarity between a new route and a query is no less than a user-specified threshold (i.e., $q_i.\theta$ in Figure 1), we say that the query matches the new route and we deliver the new route to the query. From Figure 1, we can see that q_1 and q_2 match τ_1, q_2 matches τ_2, and q_3 matches τ_3.

Our C-RSL problem is applied in dynamic spatial networks. The reason is that in many real-life scenarios, edge weights of networks (e.g., travel times, taxi fares) are changing over time (Pfoser, Tryfona, and Voisard 2006; Ding, Yu, and Qin 2008; Hua and Pei 2010). We adopt aggregate-cost measurement (i.e., the sum of (travel) costs between query locations and a route) (Shang et al. 2014; 2017b; Chen et al. 2010) to measure the similarity between a route and query locations.

The C-RSL problem aims to feed a large number of C-RSL queries with similar routes in a real-time fashion. It is challenging due to its high computation complexity. When

*Corresponding author.

[1]https://data.cityofnewyork.us/browse?q=trip

a new route is committed, we need to calculate the similarity between the new route and each query, which is very time consuming. To alleviate the computational cost, we develop Query Location Expansion Matching (QLOE) algorithm that explores the spatial network from each vertex containing query locations through network expansion (Dijkstra 1959) and retrieve the C-RSL queries that match the new route. Expansion from each vertex occurs in parallel. Our QLOE algorithm substantially reduces the time cost of processing a new route. In addition, we propose an optimization for the QLOE algorithm (i.e., QLOE+) by developing *vertex filtering* and *group expansion* techniques that are able to filter out "unqualified" queries in early stage and combine network expansion results from different vertices, respectively. Experimental study shows that QLOE+ outperforms QLOE by at least an order of magnitude regarding the efficiency of route processing.

Our contributions can be summarized as follows. First, we study a novel problem, the C-RSL problem, that continuously feeds a large number of users with routes similar to their query locations, thus targeting applications such as real-time route planning and recommendation, ridesharing, and other location-based services that have real-time demand. Second, we develop two efficient algorithms, QLOE and QLOE+, to match routes with a large number of C-RSL queries in a real-time fashion. Further, we extend our QLQE series to support top-k C-RSL problem where each query continuously maintains k most similar routes. Third, we conduct experiments on large route datasets to comprehensively study the performance of the algorithms. Our experiment results confirm the capability of QLOE+ to handle C-RSL queries over real-world route streams.

Preliminaries and Problem Formulation

This section introduces road networks, routes, the RSL query, and the C-RSL problem.

Road Networks and Routes

Following existing studies (Chen et al. 2019a; Shang et al. 2017b), we formulate road network by a connected, undirected graph $G = (V, E, F, W)$, where V is a vertex set and $E \subseteq \{\{v_i, v_j\} | v_i, v_j \in V \wedge v_i \neq v_j\}$ is an edge set. A vertex $v_i \in V$ denotes an intersection or an end point, and an edge $e = \{v_i, v_j\} \in E$ denotes a road segment connecting between v_i and v_j. Function F maps a vertex to a spatial point location with longitude and latitude and maps an edge to a polyline that represents the road segment. Function $W : (E, t) \rightarrow R$ assigns a real-valued time-dependent weight $W(e, t)$ to an edge e that denotes the travel cost of the road segment at time t. We use $lc(v_i, v_j)$ to denote the lowest cost between v_i and v_j and use $LP(v_i, v_j)$ to denote the path with the lowest cost between v_i and v_j. This modeling of road networks aligns with a number of previous studies (Chen et al. 2010; Shang et al. 2014; Chen et al. 2019a; Shang et al. 2017b). Here, a route is defined by Definition 1.

Definition 1: (*Route*) Route τ is a sequence $\langle p_1, p_2, ..., p_n \rangle$ that consists of at least 2 vertices (points), where p_i and p_{i+1} ($i \in [1, n-1]$) are adjacent vertices in V. □

RSL Query and Cost Measures

Given a query location o and a route τ, the network travel cost $c(o, \tau)$ between them is defined by Equation 1.

$$c(o, \tau) = \min_{p_i \in \tau} \{lc(o, p_i)\}, \quad (1)$$

where $lc(o, p_i)$ denotes the lowest network travel cost between o and p_i. Given a set O of query locations and a route τ, the similarity $\mathrm{Sim}(O, \tau)$ between them is defined according to aggregate travel cost (Shang et al. 2014; Chen et al. 2010; 2019a):

$$\mathrm{Sim}(O, \tau) = \sum_{o \in O} e^{-c(o,\tau)} \quad (2)$$

Based on the travel cost measures, we present the definition of RSL problem.

Definition 2: (*RSL Problem*) An RSL query $q = \{O, \theta\}$ consists of set O of query locations $\{o_1, o_2, ..., o_n\}$ and a similarity threshold θ. Given a set R of routes, the RSL query q finds a subset R' of R such that $\forall(\tau \in R')\ (\mathrm{Sim}(q.O, \tau) \geq \theta)$ and $\forall(\tau \in R \setminus R')\ (\mathrm{Sim}(q.O, \tau) < \theta)$. □

C-RSL Problem

Based on Definition 2, we formally define the C-RSL problem in Definition 3.

Definition 3: (*C-RSL Problem*) Given a set $\mathcal{Q} = \{q_1, q_2, ..., q_{|\mathcal{Q}|}\}$ of continuous RSL (C-RSL) queries and a stream of routes, the C-RSL problem aims to maintain an up-to-date result for each RSL query in $\mathcal{Q}$. □

In our applications, the typical arrival rate of travel routes in a metropolitan area (e.g., New York City) is in the scale of hundreds of thousands a day and we may serve tens of thousands of C-RSL queries at one time. We aim to develop an efficient and scalable solution to maintain the up-to-date results for a large number of C-RSL queries over a stream of routes. Routes and C-RSL queries can be easily maintained by the available memory of modern servers (Chen et al. 2015; Chen and Cong 2015). As a result, our solution is developed under in-memory setting.

Direct Route Matching

We propose a baseline algorithm, Direct Route Matching (DRM), to answer the C-RSL problem. Its high-level idea works as follows. When a user completes and commits a trip, a new travel route τ is published. We compute the similarity between each C-RSL query $q_i \in \mathcal{Q}$ and τ (i.e., $\mathrm{Sim}(q_i.O, \tau)$). If $\mathrm{Sim}(q_i.O, \tau) \geq q_i.\theta$, we deliver route τ to q_i and regard τ as a result of q_i. To compute $\mathrm{Sim}(q_i.O, \tau)$, we calculate the network travel cost between τ and each location o in $q_i.O$.

Algorithm 1 presents the pseudo code of DRM. The inputs are C-RSL query set $\mathcal{Q}$ and a new route τ. The output is a subset of *matched* C-RSL queries $\mathcal{D}$ that route τ will be delivered to. After initializing $\mathcal{D}$ (Line 1), we evaluate each RSL query $q_i \in \mathcal{Q}$ and check whether the new route τ can be a result of O_i. Specifically, we visit each location $o \in q_i.O$ and add $e^{-c(o,\tau)}$ to s based on Equation 1 (Line 5). If the

current accumulated similarity s is no less than $q_i.\theta$, we add q_i to $\mathcal{D}$ (Lines 6–7). After evaluating all $q_i \in \mathcal{Q}$, we return $\mathcal{D}$ as the result and deliver τ to each query in $\mathcal{D}$.

Complexity analysis To compute $c(o, \tau)$, we need to calculate the lowest travel cost between o and each points in τ. Because the travel cost on each edge is changing over time, it is impossible to pre-compute the lowest costs between all pairs of vertices in network. Consequently, we need to calculate the lowest cost between two vertices online. In this case, the time complexity of DRM is $O(|\mathcal{Q}| \cdot n_o \cdot |\tau| \cdot (|E| + |V| \log |V|))$, where $|\mathcal{Q}|$ is the cardinality of C-RSL queries, n_o denotes the number of locations in each C-RSL query, and $|\tau|$ is the number of points in route τ.

Algorithm 1: DRM

```
   Data: C-RSL query set Q, new route τ
   Result: matched C-RSL queries D
1  D ← ∅;
2  for each q_i in Q do
3      s ← 0;
4      for each location o in q_i.O do
5          s ← s + e^{-c(o,τ)};
6          if s ≥ q_i.θ then
7              D.add(q_i);
8  return D;
```

Query Location Expansion Matching

Overview

To reduce the high complexity of DRM, we propose a Query Location Expansion Matching (QLOE) algorithm. Initially, we generate a subset of V, V_s, where for each vertex $v \in V_s$ there exists at least one query q such that $v \in q.O$. We regard each vertex in V_s as an expansion center. For each center v, we perform network expansion (Dijkstra 1959) to explore the network and to compute the travel cost between v and the new route τ. Once we reach a vertex p such that $p \in \tau$, the expansion terminates and the travel cost between v and τ is derived. Next, we aggregate the travel cost to each query that contains v as one of its query location. The QLOE terminates when the similarity between each C-RSL query and τ are computed. Note that we define an upper bound on network cost to prune unqualified query candidates. The expansion of each vertex can be performed in parallel.

QLOE Algorithm

Pre-processing Given a C-RSL query set $\mathcal{Q}$ and the vertex set V of network G, we first need to generate a subset $V_s \subseteq V$ where each $v \in V_s$ contains at least one query location. This step requires $O(|q.O|)$ when we insert each new C-RSL query q to $\mathcal{Q}$.

Before presenting the algorithm, we first define *C-RSL tuple*, which will be used to record the visiting status and current upper bound of similarity between the C-RSL query and the new route. C-RSL tuples will be retrieved and updated during network expansion from different query locations.

Definition 4: (*C-RSL Tuple*) Let τ be a new route. A C-RSL tuple of query q is denoted by $T(q) = \langle e, n, ub \rangle$, which consists of three elements: an entry e (identifier) of query q, the number of visited query locations n in $q.O$, and the similarity upper bound ub between q and τ. Let $q.O^+ \subseteq q.O$ be a subset of $q.O$ where for each $o_i \in q.O^+$, o_i is scanned by network expansion. We compute ub via Equation 3

$$T(q).ub = |q.O| - T(q).n + \sum_{v_i \in q.O^+} e^{-c(v_i, \tau)} \quad (3)$$

□

At the beginning, we initialize a C-RSL tuple set $\mathcal{T}$ by adding an initial tuple $T(q)$ of each query q. Here, we set $T(q).n$ and $T(q).ub$ to be 0 and $|q.O|$, respectively.

Algorithm 2 presents the pseudo code of QLOE. The inputs are expansion center v, new route τ, and C-RSL tuple set $\mathcal{T}$. The output is an updated tuple set $\mathcal{T}$ that filters out unqualified tuples.[2] We first initialize the current vertex we scanned as *null* (Line 1). Next, we perform a network expansion from vertex v (Lines 2–17). Specifically, if v_c is null, it indicates that the expansion has not been started, so we assign v to v_c (Lines 3–4); otherwise, we assign the next vertex based on Dijkstra expansion to v_c (Lines 5–6). If v_c is a point of τ, we terminate the expansion from v and evaluate each query locations associated with v (Lines 7–16). For each query q of which locations contain v_c, we update its tuple $T(q)$ in $\mathcal{T}$. In particular, we add $T(q).n$ by one and update the similarity upper bound of q (i.e., $T(q).ub$) based on Definition 4 (Lines 10–11). If $T(q).ub$ is less than the similarity threshold θ defined by q, we remove tuple $T(q)$ from $\mathcal{T}$ because τ cannot be a result of q (Lines 12–13); otherwise, we update $T(q)$ to $\mathcal{T}$ (Line 15).

Note that the expansion of each vertex can be processed in parallel. After running QLOE on each vertex, the C-RSL tuples in $\mathcal{T}$ are queries that can include τ as their results.

Complexity analysis The worst-case time complexity of running QLOE on all vertices is $O((|E|+|V| \log |V|) \cdot |V| + n_o \cdot |\mathcal{Q}|)$ where n_o is the number of query locations of each query. Since the expansion from each vertex can be terminated as soon as one of the points in route τ is reached, it is unnecessary to scan all vertices in real scenario.

QLOE with Optimized Expansion

In QLOE, we need to run network expansion from each non-empty vertex, which is very time consuming. To address this problem, we develop an optimized network expansion approach, QLOE+, to filtering unnecessary expansions. Specifically, during the expansion from vertex v to new route τ, we record the lowest cost between v and every vertex we scanned. To avoid a vertex being scanned repeatedly by expansions from different vertices, we maintain global vertex map $\mathcal{M}$ to record scanned vertices. Further, we define the concept of *vertex filtering condition* (Theorem 1) to filter "unqualified" queries associated with a particular vertex.

[2] We say that $T(q)$ is an unqualified tuple if q cannot match τ

Algorithm 2: QLOE

```
Data: Vertex v, new route τ, C-RSL tuples 𝒯
Result: Updated C-RSL tuples 𝒯
1  v_c ← null;
2  do
3      if v_c is null then
4          v_c ← v;
5      else
6          v_c ←DijkstraExpansion(v).next();
7      if v_c ∈ τ then
8          for each query q that contains v do
9              if T(q) exists in 𝒯 then
10                 T(q).n ← T(q).n + 1;
11                 T(q).ub ← T(q).ub − 1 + e^{−lc(v,v_c)};
12                 if T(q).ub < q.θ then
13                     𝒯.remove(T(q));
14                 else
15                     Update T(q) in 𝒯;
16         break;
17 while DijkstraExpansion(v).hasNext();
```

Theorem 1: Let Q_v be a set of queries associated with vertex v and v_c be the current vertex being visited by network expansion. All queries in Q_v can be safely filtered if the following inequality is satisfied.

$$\max_{q_i \in Q_v} \{|q_i.O|\} - 1 + \exp(-lc(v, v_c)) < \min_{q_i \in Q_v} \{q_i.\theta\}. \quad (4)$$

Proof: Because the expansion terminates when we reach the new route τ, we have $lc(v, v_c) \leq \min_{v_i \in \tau}\{lc(v, v_i)\}$. Based on Equations 1 and 2, we have $lc(v, v_c) \leq c(v, \tau)$. Thus, if Inequality 4 is satisfied, we can deduce that $\forall(q_i \in Q_v)$ $(\text{Sim}(q_i.O, \tau) \leq q_i.\theta)$. We complete the proof.

Algorithm 3 presents the pseudo code of QLOE+. First, we initialize H, v_c, and Q_v (Lines 1–3). Here, H is used to record scanned vertices v_c and the lowest cost between v and v_c (i.e., $\langle v_c, lc(v, v_c)\rangle$) in network expansion from v. Then we perform the network expansion. When we reach a new vertex v_c, we check if query set Q_v satisfies vertex filtering condition (Theorem 1). If so, we add v to $\mathcal{M}$, remove all tuples of queries in Q_v from $\mathcal{T}$, and terminate the expansion (Lines 9–12). If there exists at least one query associated with v_c and it has never been scanned, we add $\langle v_c, lc(v, v_c)\rangle$ to H (Line 14). If $v_c \in \tau$, we evaluate queries associated with each v_i in H. Specifically, we first add v_i to $\mathcal{M}$ denoting that v_i has been scanned (Line 17). Then we update the C-RSL tuples of each query q that contains v_i (Lines 18–25). The steps are the same as those in QLOE (Algorithm 2).

The expansion of each vertex can be processed in parallel. Recall that QLOE+ maintains global vertex map $\mathcal{M}$ to record scanned vertices. We do not need to perform expansion from a scanned vertex. Thus, when an expansion thread is completed, we randomly pick a vertex $v \in V \setminus \mathcal{M}$ and perform expansion from v. When $\mathcal{M} = V$, which means that all vertices have been scanned, we return $\mathcal{T}$ as the result.

Algorithm 3: QLOE+

```
Data: Vertex v, new route τ, C-RSL tuples 𝒯, vertex map ℳ
Result: Updated C-RSL tuples 𝒯, updated vertex map ℳ
1  H ← ∅;
2  v_c ← null;
3  Q_v ← set of queries associated with v;
4  do
5      if v_c is null then
6          v_c ← v;
7      else
8          v_c ←DijkstraExpansion(v).next();
9          if Q_v satisfies vertex filtering condition then
10             ℳ.add(v);
11             𝒯.removeAll(T(Q_v));
12             break;
13     if there exists a query that contains v_c then
14         H.add(⟨v_c, lc(v, v_c)⟩);
15         if v_c ∈ τ then
16             for each ⟨v_i, ·⟩ ∈ H do
17                 ℳ.add(v_i);
18                 for each query q that contains v_i do
19                     if T(q) exists in 𝒯 then
20                         T(q).n ← T(q).n + 1;
21                         T(q).ub ←
                               T(q).ub − 1 + e^{−lc(v_i,v_c)};
22                         if T(q).ub < q.θ then
23                             𝒯.remove(T(q));
24                         else
25                             Update T(q) in 𝒯;
26                 break;
27 while DijkstraExpansion(v).hasNext();
```

Extension of Processing Top-k C-RSL Queries

This section discuss how to extend our QLOE series to support top-k continuous RSL query, which is defined by Definition 5.

Definition 5: (*Top-k C-RSL Problem*) A top-k C-RSL query $q = \langle O, k\rangle$ consists of set O of query locations $\{o_1, o_2, ..., o_n\}$ and the number of results k. Given a set R of routes, the top-k RSL query q finds k routes that has the highest similarities to O. Given a set $\mathcal{Q} = \{q_1, q_2, ..., q_{|\mathcal{Q}|}\}$ of top-k C-RSL queries and a stream of routes, the C-RSL problem aims to maintain a top-k result for each query in $\mathcal{Q}$. □

Recall that for threshold-based C-RSL problem, the route matching is solely based on a static similarity threshold $q.\theta$ specified by each query q. While for top-k C-RSL problem, we need to maintain a dynamic set of k most similar routes. To solve the problem, for each top-k C-RSL query q we regard the similarity between q and the $q.\tau_k$ (i.e., the k-th result route maintained by q) as the "similar-

ity threshold". Specifically, we a new route τ_n arrives, for each query q we compute the similarity between $q.O$ and τ. If $\mathrm{Sim}(q.O, \tau_n) > \mathrm{Sim}(q.O, q.\tau_k)$, we replace τ_k by τ_n and update q's result set; Otherwise, we discard τ_n. Note that when the result of q is updated, we also need to update the "similarity threshold" because the original $q.\tau_k$ is changed. As a result, both QLOE and QLOE+ can support the processing of top-k C-RSL queries by regarding $q.\tau_k$ in top-k C-RSL as $q.\theta$ in threshold-based C-RSL.

Experimental Study

This section reports our experiments conducted on real road networks and route data sets. The results offer insight into the efficiency and scalability of the proposed algorithms.

Experiment Settings

Data sets and query generation We use two datasets in our experiments: Beijing Road Network (BJR) and the New York Road Network (NYR)[3]. BJR consists of a road network of Beijing and a real taxi trajectory data set collected by the T-drive project (Yuan et al. 2013). NYR consists of a road network of New York City and a taxi trip data set from New York[3], which only contains pick-up and drop-off locations of a taxis. To generate the route of a taxi trip by deriving the shortest path from the pick-up location to the drop-off location. To simulate dynamic edge weights on road networks, we let $d\%$ of edges change by a random ratio ranging from 0.8x to 1.2x each time we complete the processing of a new route (Shang et al. 2016). Note that d is a parameter set as 1. The efficiency of our methods is not influenced by d.

A C-RSL query location set $q.O$ is generated as follows: First, we sample 100K routes and calculate the number of routes passing each vertex. Next, we pick vertices based on the following probability formulation:

$$P(v) = \frac{n_v}{\sum_{\tau \in S} |\tau|},$$

where $P(v)$ is the probability that v is picked, S is a set of sampled routes, and n_v denotes the number of routes passing v.

Implementations The road network graphs, routes, and C-RSL queries are memory resident. All algorithms are implemented in Java and run on a server with two Intel® Xeon® Processors Gold 5120 (2.20GHz) and 64GB RAM. The performance metric is runtime of processing a new route (i.e., finding a subset of queries that match a new route). We evaluate the following three methods: DRM, QLOE, and QLOE+. To enhance the efficiency of DRM, we let the computation of similarity between a new route and each query be processed in parallel (Algorithm 1, Lines 3–7). The parameter settings are listed in Table 1.

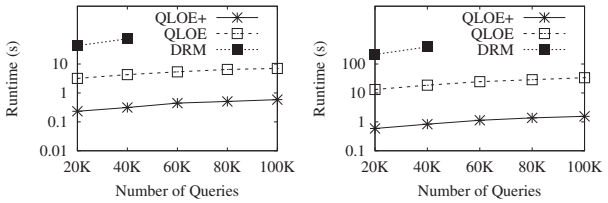
Experimental Results

Effect of the number of queries Figure 2 presents the performance of the algorithms when varying the number of C-RSL queries from 20K to 100K. We can see that the time cost of processing a new route increases for all three methods when we increase the number of queries. In particular, QLOE outperforms DRM by approximately an or-

[3]https://publish.illinois.edu/dbwork/open-data/

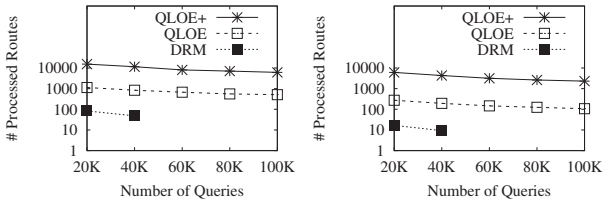
Table 1: Parameter Settings

	BJR	**NYR**
Number of queries	20,000–100,000 / default 20,000	20,000–100,000 / default 20,000
Cardinality of query location set $\lvert q.O\rvert$	3–7 / default 5	3–7 / default 5
Similarity threshold θ ($\times\lvert q.O\rvert$)	0.80–0.95 / default random	0.80–0.95 / default random
Number of results k	5–30 / default 10	5–30 / default 10
Thread count m	16–48 / default 48	16–48 / default 48



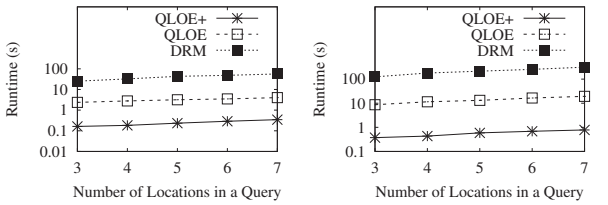
(a) BJR (b) NYR

Figure 2: Effect of the number of queries (runtime)



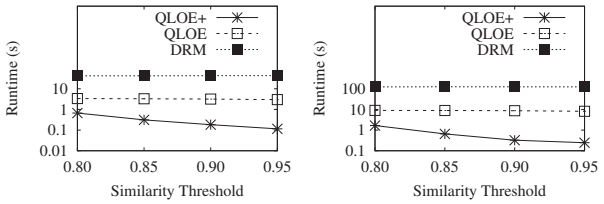
(a) BJR (b) NYR

Figure 3: Effect of the number of queries (arrival rates)



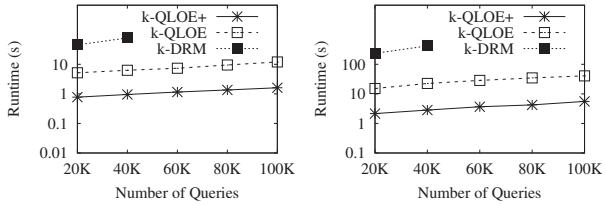
(a) BJR (b) NYR

Figure 4: Effect of the number of locations in a query



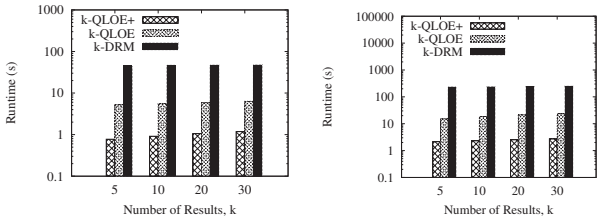
(a) BJR (b) NYR

Figure 5: Effect of similarity threshold, θ



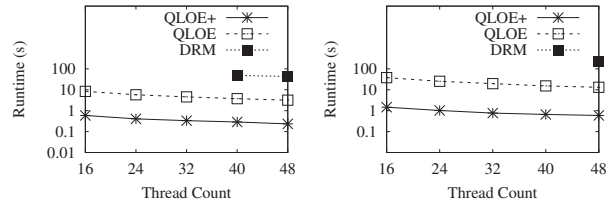
(a) BJR (b) NYR

Figure 6: Effect of the number of locations in a query (top-k)



(a) BJR (b) NYR

Figure 7: Effect of the number of results



(a) BJR (b) NYR

Figure 8: Effect of thread counts

der of magnitude for both BJR and NYR. Such significant performance discrepancy can be explained as follows: (1) The time complexity of QLOE is lower than that of DRM; (2) For QLOE, each expansion thread completes as soon as one of the point (vertex) of new route τ is scanned, which substantially improves the efficiency of route processing. In addition, compared with QLOE, QLOE+ further improves the route processing efficiency by at least an order of magnitude. Such conspicuous performance improvement confirms the effectiveness of our vertex filtering technique and optimized network expansion.

Figure 3 reports the maximum arrival rate (number of routes per hour) of the route stream that can be supported by each method given the number of subscriptions varying from 20K to 100K. We can see that when we have 20K queries, QLOE+ can support the data stream with the rates of 15K routes/h and 6K routes/h on BJR and NYR, respectively. In contrast, DRM can only support the stream with the rates of 83 routes/h and 16 routes/h on BJR and NYR, respectively. Even if we increase the number of queries to 100K, QLOE+ can still support the data stream with the rates of 6.1K routes/h and 2.2K routes/h on BJR and NYR, respectively, which are 12x and 21x better than the performances of QLOE on BJR and NYR, respectively. In the real-world scenario, the average route arrival rate in NYC is $\sim$15K/h[4]. Theoretically, QLOE+ is able to handle 100K C-RSL queries over the real route data stream if we deploy $\sim$7 modern servers. In contrast, QLOE requires more than 140 servers to handle the real route data stream in NYC.

Effect of the number of locations in a query $|q.O|$ Figure 4 shows the effect of varying $|q.O|$ on the efficiency of the algorithms. A larger $|q.O|$ implies: (1) A larger number of vertices to be expanded and evaluated for QLOE and QLOE+; (2) A larger number shortest path computations occur for DRM. Thus, when we increase the number of locations in C-RSL query, the runtime of route processing increases for all methods.

Effect of similarity threshold θ This set of experiments investigates the effect of similarity threshold θ. Figure 5 shows the results when we vary the similarity threshold θ. We can see that the performance of DRM and QLOE is consistent as we vary θ from 0.8 to 0.95. However, the runtime of QLOE+ exhibits a decreasing trend as we increase the value of θ. The reason is that the value of θ has little influence on the number of shortest path computations and search space. However, the vertex filtering power in QLOE+ becomes stronger and more queries will be filtered out when we increase θ. As a result, the efficiency of QLOE+ improves substantially when we increase θ.

Extension of processing top-k C-RSL queries Figure 6 shows the performance of the three methods for processing top-k C-RSL queries when we vary the number of queries from 20K to 100K. Compared with the performance of processing threshold-based C-RSL queries, we observe a increment of time cost for processing top-k C-RSL queries. In particular, the runtime of DRM, QLOE, and QLOE+ are increased by factors of $\sim$1.1, 1.2–1.6, and 2.5–4.0, respectively. The significant performance contrast regarding QLOE+ can be explained by the fact that QLOE+ requires extra time cost for maintaining the dynamic value of the similarity between $q.O$ and $q.\tau_k$ for each query q, which may lower the effectiveness of the vertex filtering condition.

Figure 7 presents the effect of the number of result routes maintained by each query. We can see that the performance

[4]https://data.cityofnewyork.us/browse?q=trips

of DRM remains consistent as we vary the value of k. As for QLOE series, we observe a slight increasing trend regarding the time cost as we increase k. The reason is that higher value of k is likely to lower the value of the similarity between $q.O$ and $q.\tau_k$ for each query q. Therefore, each time a new route arrives, the average number of queries that have their result set updated will increase, which may increase the time cost of route matching. Nevertheless, even though we increase k from 5 to 30, the average runtime of QLOE and QLOE+ only increases by a factor of $\sim$1.2 and $\sim$1.4, respectively.

Effect of thread counts We study the effect of thread count on the efficiency of the algorithms. The results in Figure 8 show that QLOE+ consistently outperforms QLOE and DRM by at least an order of magnitude and two orders of magnitude, respectively, when we vary the number of threads from 16 to 48.

Related Work

Route search based on locations Route search based on locations aims at finding routes that have the highest relevances to query arguments (Frentzos, Gratsias, and Theodoridis 2007; Zheng et al. 2013; Shang et al. 2012; Zheng et al. 2016; Shang et al. 2017a; 2019). Existing studies define different ranking functions to measure the relevancy between query locations and routes. Specifically, ranking functions may consider spatial (Chen et al. 2010), temporal (Shang et al. 2014), textual (Shang et al. 2012; Zheng et al. 2013), and density elements. However, existing studies regard routes or trajectories as a collection of static data and their query is performed in one-time fashion. In contrast, the C-RSL query is continuous and the routes are modeled as data streams instead of a collection of data.

Route similarity joins This line of research focuses on the problem of finding route pairs or trajectory pairs with similarities higher than a threshold (Ding, Trajcevski, and Scheuermann 2008; Ray et al. 2015; Chen and Patel 2009; Bakalov et al. 2005; Bakalov and Tsotras 2006; Ta et al. 2017; Shang et al. 2017b; 2018). Existing studies solve the problem of trajectory similarity join by the following steps: (1) similarity definition step and (2) join processing step. In similarity definition step, a ranking function that evaluates the similarity between two trajectories is defined. Basically, the function takes spatial proximity and temporal relevancy into account. For join processing step, efficient searching algorithm is proposed to find all trajectory pairs whose similarities are above a user-defined threshold. However, existing studies on route similarity joins take a collection of trajectories or routes as input while our problem takes a stream of routes as input. As a result, the aforementioned proposals cannot be applied to solve our C-RSL problem.

Location-based continuous query Different from "one-time" queries that find items based on a snapshot of a database, continuous queries receive items that satisfy their query predicates over data streams in a real-time fashion (Chen et al. 2019b). A host of studies exist that aim at developing query indexing structures and streaming item processing algorithms to handle a large number of location-based continuous queries over data streams (Li et al. 2013; Guo et al. 2015; Wang et al. 2015; Chen et al. 2018; Xu et al. 2017; Yang et al. 2019). The data handled by these studies is a stream of individual geo-tagged items (e.g., geo-tagged tweets). In contrast, the input data of the C-RSL problem is a stream of location sequences, which calls for different purposeful query indices and matching algorithms.

Conclusions

We propose and study C-RSL problem to enable continuous route-search-by-locations for a large number of users over route data streams. We develop two parallel route matching algorithms: Query Location Expansion Matching (QLOE) and QLOE with optimized expansion techniques (QLOE+), which substantially reduce the time complexity of answering the C-RSL problem in comparison to the baseline. Extensive experiment with real data demonstrates that QLOE and QLOE+ are capable of achieving high efficiency and scalability in answering C-RSL problem.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61932004).

References

Bakalov, P., and Tsotras, V. J. 2006. Continuous spatiotemporal trajectory joins. In *GSN*, 109–128.

Bakalov, P.; Hadjieleftheriou, M.; Keogh, E. J.; and Tsotras, V. J. 2005. Efficient trajectory joins using symbolic representations. In *MDM*, 86–93.

Chen, L., and Cong, G. 2015. Diversity-aware top-k publish/subscribe for text stream. In *SIGMOD*, 347–362.

Chen, Y., and Patel, J. M. 2009. Design and evaluation of trajectory join algorithms. In *ACM-GIS*, 266–275.

Chen, Z.; Shen, H. T.; Zhou, X.; Zheng, Y.; and Xie, X. 2010. Searching trajectories by locations: an efficiency study. In *SIGMOD*, 255–266.

Chen, L.; Cong, G.; Cao, X.; and Tan, K. 2015. Temporal spatial-keyword top-k publish/subscribe. In *ICDE*, 255–266.

Chen, L.; Shang, S.; Zhang, Z.; Cao, X.; Jensen, C. S.; and Kalnis, P. 2018. Location-aware top-k term publish/subscribe. In *ICDE*, 749–760.

Chen, L.; Shang, S.; Jensen, C. S.; Yao, B.; Zhang, Z.; and Shao, L. 2019a. Effective and efficient reuse of past travel behavior for route recommendation. In *KDD*, 488–498.

Chen, L.; Shang, S.; Yang, C.; and Li, J. 2019b. Spatial keyword search: a survey. *GeoInformatica*.

Chen, L.; Cong, G.; and Cao, X. 2013. An efficient query indexing mechanism for filtering geo-textual data. In *SIGMOD*, 749–760.

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerische Math* 1:269–271.

Ding, H.; Trajcevski, G.; and Scheuermann, P. 2008. Efficient similarity join of large sets of moving object trajectories. In *TIME*, 79–87.

Ding, B.; Yu, J. X.; and Qin, L. 2008. Finding time-dependent shortest paths over large graphs. In *EDBT*, 205–216.

Frentzos, E.; Gratsias, K.; and Theodoridis, Y. 2007. Index-based most similar trajectory search. In *ICDE*, 816–825.

Guo, L.; Zhang, D.; Li, G.; Tan, K.; and Bao, Z. 2015. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *SIGMOD*, 843–857.

Hua, M., and Pei, J. 2010. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, 347–358.

Li, G.; Wang, Y.; Wang, T.; and Feng, J. 2013. Location-aware publish/subscribe. In *KDD*, 802–810.

Pfoser, D.; Tryfona, N.; and Voisard, A. 2006. Dynamic travel time maps - enabling efficient navigation. In *SSDBM*, 369–378.

Ray, S.; Brown, A. D.; Koudas, N.; Blanco, R.; and Goel, A. K. 2015. Parallel in-memory trajectory-based spatiotemporal topological join. In *Big Data*, 361–370. IEEE.

Shang, S.; Ding, R.; Yuan, B.; Xie, K.; Zheng, K.; and Kalnis, P. 2012. User oriented trajectory search for trip recommendation. In *EDBT*, 156–167.

Shang, S.; Ding, R.; Zheng, K.; Jensen, C. S.; Kalnis, P.; and Zhou, X. 2014. Personalized trajectory matching in spatial networks. *VLDB J.* 23(3):449–468.

Shang, S.; Chen, L.; Wei, Z.; Guo, D.; and Wen, J. 2016. Dynamic shortest path monitoring in spatial networks. *J. Comput. Sci. Technol.* 31(4):637–648.

Shang, S.; Chen, L.; Jensen, C. S.; Wen, J.; and Kalnis, P. 2017a. Searching trajectories by regions of interest. *IEEE Trans. Knowl. Data Eng.* 29(7):1549–1562.

Shang, S.; Chen, L.; Wei, Z.; Jensen, C. S.; Zheng, K.; and Kalnis, P. 2017b. Trajectory similarity join in spatial networks. *PVLDB* 10(11):1178–1189.

Shang, S.; Chen, L.; Wei, Z.; Jensen, C. S.; Zheng, K.; and Kalnis, P. 2018. Parallel trajectory similarity joins in spatial networks. *VLDB J.* 27(3):395–420.

Shang, S.; Chen, L.; Zheng, K.; Jensen, C. S.; Wei, Z.; and Kalnis, P. 2019. Parallel trajectory-to-location join. *IEEE Trans. Knowl. Data Eng.* 31(6):1194–1207.

Ta, N.; Li, G.; Xie, Y.; Li, C.; Hao, S.; and Feng, J. 2017. Signature-based trajectory similarity join. *IEEE Trans. Knowl. Data Eng.* 29(4):870–883.

Wang, X.; Zhang, Y.; Zhang, W.; Lin, X.; and Wang, W. 2015. Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In *ICDE*, 1107–1118.

Xu, Y.; Chen, L.; Yao, B.; Shang, S.; Zhu, S.; Zheng, K.; and Li, F. 2017. Location-based top-k term querying over sliding window. In *WISE*, 299–314.

Yang, C.; Chen, L.; Shang, S.; Zhu, F.; Liu, L.; and Shao, L. 2019. Toward efficient navigation of massive-scale geo-textual streams. In *IJCAI*, 4838–4845.

Yuan, J.; Zheng, Y.; Xie, X.; and Sun, G. 2013. T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE Trans. Knowl. Data Eng.* 25(1):220–232.

Zheng, K.; Shang, S.; Yuan, N. J.; and Yang, Y. 2013. Towards efficient search for activity trajectories. In *ICDE*, 230–241.

Zheng, K.; Zheng, B.; Xu, J.; Liu, G.; Liu, A.; and Li, Z. 2016. Popularity-aware spatial keyword search on activity trajectories. *World Wide Web* 19(6):1–25, online first.