Karlsruher Institut für Technologie

Flexible User-Friendly Trip Planning Queries

Bachelorarbeit
von

cand. inform. Violina Zhekova

an der Fakultät für Informatik

Erstgutachter:	Dr. Ing. Martin Schäler
Zweitgutachter:	
Betreuender Mitarbeiter:	M.-Sc. Saeed Taghizadeh

Bearbeitungszeit: 01. November 2018 – 14. April 2019

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 26. Februar 2019

Abstract

Trip planning queries are often from the type Sequenced route Queries (SRQ), a form of nearest neighbor queries, which define a starting point and a list of categories, given by the user. This type of queries are gaining significant interest, because of advances in location based mobile services and they are also of great importance in developing robust systems, where crisis management is of utter importance.

Existing approaches strive to find a best route, based on length, duration or other prime factors, passing through multiple location, called points of interest (PoIs), and they match the route perfectly. However, users may be also interested in other qualities of the route, such as the relationship among sequence points, hierarchy, order and priority of the PoIs. Therefore, in this thesis I introduce a set of operators, which the users may be interested in applying to SRQ, and propose approaches to designing and implementing some of the operators. The implementation considers metric spaces, as these are mostly relevant to the user, when working with road networks in real-life maps.

Contents

Chapter 1

Introduction

A sequenced route query is defined as finding the shortest path from a starting point towards a possible destination, passing through multiple locations, defined by their category type. There has been significant research and proposed approaches on the topic, but there is not a developed query language to answer this types of queries. The work in this thesis has been focused on researching the topic of sequenced route queries and designing a language to enable the user to express his need in the form of a user query in a flexible manner, such as applying different constraints on the route to be found.

Example: Suppose that a user is planning a trip to town: he first wants to go to a restaurant for lunch, then he wants to stop by a bank, then he meets a friend in the shopping mall and after that he plans to have a dinner at a restaurant. In this specific scenario, the user wants to express his wish for the restaurant to be the same.

The specific constrains such as the equality in the given example above are proposed in the thesis as operators on the query. Existing approaches have been used to transform the complex user query and changes to the approaches have been made in order to retrieve a desired result.

The completeness of the operators stems from

Problem definition: We have a starting point sp and a category sequence $M = (c_1, c_2, ..., c_n)$, which constitutes the query, defined by the user. The constraints for this query can be applied as operators. For this query a route $(r_1, r_2, ..., r_n)$, defined as a sequence of PoIs, is calculated.

Construction: The graph is constructed using the Neo4j graph database as seen in 1.1 The crossroads are defined as nodes, labeled CROSSROAD, with attributes id, longitute and latitude, the roads are mapped as relationships, labeled ROAD, with attribute distance. A PoI is mapped to the nearest CROSSROAD, using a relationship HAS-POI, with attributes id and a list of possible category types it belongs to. [1]

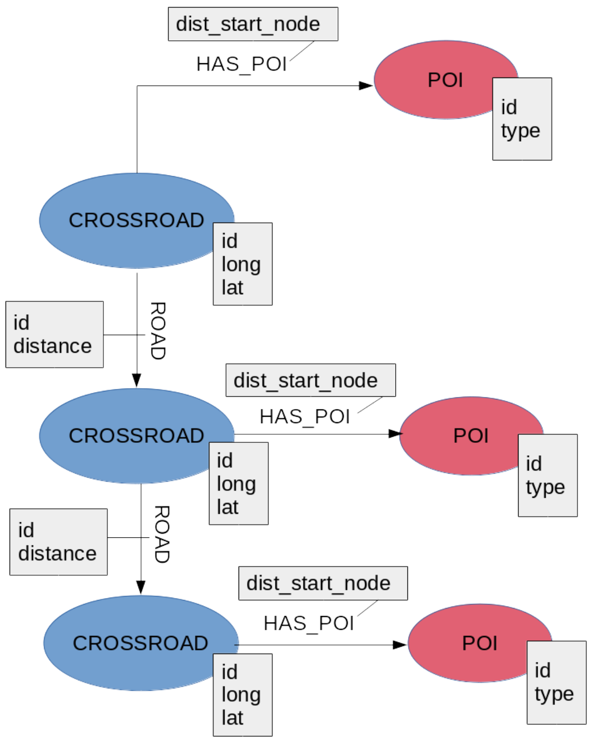

Figure 1.1: Neo4j Model [1]

The map used for implementation and testing is the road network of Berlin, with 428769 nodes, 504229 roads, 5548 PoIs and 7 category types: restaurant, coffee shop, atms/banks, movie theaters, pharmacies, pubs/bars, gas stations.

The remainder of the thesis is organized as follows: First I review the related work that has been done on the topic of SRQ in Section 2. In Section 3 I cover the proposed operators and go into details on some of them in three separate sections for each of them: Design, Implementation and Evaluation. Finally, I conclude the thesis by summing up the progress made on the subject and discuss future work.

Chapter 2

Related Work

In this section I would like to review some existing research, related to the topic of this thesis. Sequenced route queries have been extensively researched and different algorithms that optimize the problem and address different use scenarios have been developed. Usually, existing approaches differentiate between vector and metric spaces, considering the Euclidean distance between geographic points or the real-life road-network-based distances accordingly. Some algorithms are focused on returning a single optimal route, where the PoIs match the given categories in the category sequence perfectly, whereas others consider semantic hierarchy or multiple route factors such as rating, distance and category weights.

In *The Optimal Sequenced Route* the researchers propose two effective algorithms for solving the sequenced route query problem. They first elaborate on why a classic shortest path algorithm such as Dijkstra would be impractical for real-life scenarios and then go on to propose the LORD (Light Optimal Route Discoverer) and R-LORD algorithm, which uses a R-tree, which are Dijkstra-based and made for vector spaces and the PNE (Progressive Neighbor Exploration) algorithm, which emplys the nearest neighbour search and is designed specifically for metric spaces. Both of their proposed algorithms calculate a perfect route and only return one optimal route (while modification of the PNE algorithm also allow for finding k optimal routes), significantly outperforming Dijkstra's algorithm. [5]

A different approach to the SRQ, designed for metric spaces, is proposed in *Sequenced Route Query with Semantic Hierarchy*. The authors suggest a Skyline based algorithm, called bulk SkySR (BSSR), which searches for all preferred routes to users by extending the shortest route search with the semantic similarity of PoIs' categories. This approach expects a category tree, representing the semantic hierarchy of categories, and applies the Skyline concept, which is searching for routes that are not worse than any other routes in terms of their scores, to the route length and semantic similarity, also known as the route scores. The BSSR algorithm also exploits the branch-and-bound concept by searching for routes simultaneously to reduce the search space. [6]

Another research article proposes the Personalized and Sequenced Route (PSR) Query, which considers both personalization and sequenced constraints. The approach takes into account multiple factors of a route, such as distance rating and associates different weight with each PoI category and a distance weight. The framework designed to obtain one optimal route consists of three phases: guessing, crossover and refinement, and is focused on spatial databases. [3]

In *In-Route Skyline Querying for Location-Based Services* queries are issued by user moving along a routes towards destinations (PoIs), also defined as query points. The movement of the user is constrained to a road network and the travel distance is considered. In-route queries know the destination and current location of the user, which dynamically changes, and the anticipated route towards the end-point. Users can apply weights to several spatially-related criteria, when deciding on PoIs to visit next, such as the total distance difference, known as detour, and the relative distance of the current data point. [4]

An article *Sequenced Route Queries: Getting Things Done on the Way Back Home* suggest speedup techniques for sequenced route queries. A contraction hierarchy is proposed for preprocessing results for faster retrieval of answers by shortest path queries in road networks. The second technique uses the distance sensitivity of routes ("most queries are of a local kind"), which it bases on users' typical behavior. In this approach, one optimal route is returned, but queries where the order of PoIs is not necessarily fixed are possible as long as the number of PoIs remains moderate. Also, constraints on the order of visited PoIs can be made, e.g. visiting a restaurant before a shopping center. [2]

Chapter 3

Operators

In this chapter the proposed operators are covered in terms of their design, implementation and evaluation.

Necessity: some of the PoIs in the route can be missing
Conjunction, disjunction and negation: applied to some of the user-specified categories in the sequence
Order: some of the PoIs in the route must be in the given order
Hops between PoIs: defined number of PoIs between the given categories
Perfection: some of the PoIs in the route must match the user-specified category perfectly

3.1 Equality operator

The equality operator is based on the need to express that some PoIs in the SRQ of the same category can or should be equal, as given in the example in Chapter 1 1

3.1.1 Design

The equality operator is designed using the PNE approach from [5]. It uses the progressive neighbour explorator as its base to upgrade on and extends it with a heuristic approach to shrink the search space.

3.1.2 Implementation

3.1.3 Evaluation

Algorithm 1: equalityOperator

```
Input  : $Query(sp, M = (c_1, c_2, ..., c_n)), Equal(i, j)$
Output: $Route = (r_1, r_2, ..., r_n)$
$initialize\ heap$;
$initialize\ UB$;
$optimalRoute =$ PNE($Query$);
if $optimalRoute[i] = optimalRoute[j]$ then
    optimal route has been found;
    return $optimalRoute$;
else
    dummyPSR();
    modifiedPNE();
end
```

Algorithm 2: `modifiedPNE`

```
foreach $r_i$ in $U_{M_i}$ do // Checking the upper bound for
    every $r_i$ in the category set $U_{M_i}$
if travelDistance(($r_{i-1}, r_i$)) < $UB$ then
    place the new PSR $(r_{i-1}, r_i)$ on the $heap$;
    fetch a $PSR$ from the $heap$;
    switch $l = length(PSR)$ do
        case $l < m - 2$ do Finding PSRs before $r_j$
            case1();
        end
        case $l = m - 1$ do Finding PSR containing $r_j$
            case2();
        end
        case $l = m$ do Finding PSR after $r_j$
            case3();
        end
        case $l = m + 1$ do Finding PSR containing $r_{j+1}$
            case4();
        end
        case $l = m + 2$ do Finding optimal PSR $r_{i-1}, ..., r_{j+1}$
            $secondPSR = (r_{i-1}, ..., r_{j+1})$ ;    // Second part of
             the route
            $thirdPSR = PNE(r_{j+1}, (c_{j+2}, ..., c_n))$ ; // Third part
             of the route
            return $concatinate(firstPSR, secondPSR, thirdPSR)$;
        end
    end
end
;
```

Procedure dummyPSR

```
1  // Dividing the route in three parts
   $(r_1, ..., r_i), (r_i, ..., r_j), (r_j, ..., r_n)$ to be found
   consequetively
2  // Creating a dummy PSR (partial sequence route)
   from the found optimal route and calculating
   an upper bound
3  $firstPSR = (r_1, r_2, ..., r_{i-1})$ ;   // First part of the route
4  $dummyPSR \leftarrow$ remove $r_1$ to $r_{i-2}$ and $r_{j+2}$ to $r_n$ from $optimalRoute$;
5  $dummyPSR(j) = optimalRoute[i]$;
6  $r_{j+1} =$ nextNearestNeigbour($r_i, U_{M_{j+1}}$);
7  $dummyPSR = (r_{i-1}, r_i, ..., r_i, r_{j+1})$;
8  $UB =$ travelDistance($dummyPSR$);
9  remove $optimalRoute$ from the $heap$;
10 place $dummyPSR$ on the $heap$;
```

Procedure caseOne

```
1  a) nextNearestNeigbour($r_k, U_{M_{k+1}}$);
2  update $PSR$ to contain $r_{k+1}$;
3  if travelDistance($PSR$) $< UB$ then
4      update $UB$;
5      place $PSR$ on the $heap$;
6  else
7      break;
8  end
9  b) nextNearestNeigbour($r_{k-1}, U_{M_k}$);
10 update $PSR$;
11 if travelDistance($PSR$) $< UB$ then
12     update $UB$;
13     place $PSR$ on the $heap$;
14 else
15     break;
16 end
```

Procedure caseTwo

```
a) travelDistance(($r_{j-1}, r_i$));
update $PSR$ to contain $r_i$ in the place of $r_j$;
if travelDistance($PSR$) $< UB$ then
    update $UB$;
    place $PSR$ on the $heap$;
else
    break;
end
b) nextNearestNeigbour($r_{j-2}, U_{M_{j-1}}$);
update $PSR$;
if travelDistance($PSR$) $< UB$ then
    update $UB$;
    place $PSR$ on the $heap$;
else
    break;
end
```

Procedure caseThree

```
a) nextNearestNeigbour($r_j, U_{M_{j+1}}$);
update $PSR$ to contain $r_{j+1}$;
if travelDistance($PSR$) $< UB$ then
    update $UB$;
    place $PSR$ on the $heap$;
else
    break;
end
b) travelDistance(($r_{j-1}, r_i$));
update $PSR$ to contain $r_i$ in the place of $r_j$;
if travelDistance($PSR$) $< UB$ then
    update $UB$;
    place $PSR$ on the $heap$;
else
    break;
end
```

Procedure caseFour

```
nextNearestNeigbour($r_i, U_{M_{j+1}}$);
update $PSR$ to contain $r_{j+1}$;
if travelDistance($PSR$) $< UB$ then
    update $UB$;
    // Keeping only one route $(r_{i-1}, ..., r_{j+1})$ on the heap
    place $PSR$ on the heap and remove longer PSR;
else
    break;
end
```

Algorithm 3: PNE

```
// Incrementally create the set of candidate routes for $Query(sp, M)$ from starting point $sp$ towards PoI set $U_{M_m}$
// Candidate routes are stored in a heap sorted by length of the routes
// At each iteration of PNE a $PSR$ (partial sequenced route) is fetched and examined based on its length
// Trimming: There must be only one candidate SR on the heap
switch $l = length(PSR)$ do
    case $l = m$ do
        $PSR$ is the optimal route;
    end
    case $l \neq m$ do
        a) nextNearestNeigbour($r_{|}PSR|, U_{m_{|PSR|+1}}$);
        update $PSR$ and put it back on the heap;
        b) nextNearestNeigbour($r_{|PSR|-1}, U_{m_{|PSR|}}$);
        generate a new $PSR$ and place it on the heap;
    end
end
```

Chapter 4

Conclusion

Bibliography

[1] Eva Giannatou. https://github.com/evagian/California-road-network-NEO4J-CYPHER-graph-and-queries.

[2] Xuegang Huang and Christian S. Jensen. In-route skyline querying for location-based services. Department of Computer Science, Aalborg University, Denmark.

[3] Jiajie Xu4 Zhiming Ding Jian Dai, Chengfei Liu. On personalized and sequenced route planning. Institute of Software, Chinese Academy of Sciences, Beijing, China, University of Chinese Academy of Sciences, Beijing, China, Department of Computer Science and Software Engineering, School of Software and Electrical Engineering, Faculty of Science, Engineering and Technology, Swinburne University of Technology, Melbourne, Australia, School of Computer Science and Technology, Soochow University, Suzhou, China, School of Computer Science, Beijing University of Technology, Beijing , China.

[4] Stefan Funke Jochen Eisner. Sequenced route queries: Getting things done on the way back home. Universitat Stuttgart, Institut fur Formale Methoden der Informatik.

[5] Mehdi Sharifzadeh and Cyrus Shahabi Mohammad Kolahdouzan. The optimal sequenced route query. Computer Science Department, University of Southern California.

[6] Yasuhiro Fujiwara Makoto Onizuka Yuya Sasaki, Yoshiharu Ishikawa. Sequenced route query with semantic hierarchy. Graduate School of Information Science and Technology, Osaka University, Osaka, Japan, Graduate School of Information Science, Nagoya University, Nagoya, Japan, NTT Software Innovation Center, Tokyo, Japan.