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 7. April 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.

Todo: main conclusion

Contents

Chapter 1

Introduction

1.1 Motivation

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, because he may prefer a route where the equality of the two restaurant PoIs is more important to him than the length of the route.

With existing approaches, the user may get the shortest route [5] or all routes that satisfy the semantic similarity and length conditions equally [6], but that does not guarantee the equality of the two restaurant PoIs. Also finding k optimal routes answering the user's SRQ and then filtering out the routes where the two PoIs of type restaurant are equal has proven to not always generate a result, which is why in this thesis an optimal approach is presented.

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.

1.2 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.

Graph model: The road network or also known as spatial network is modeled as weighted graph where the crossroads are represented by nodes and roads are represented by the edges connecting the nodes. The weights on the edges in this specific research problem are the distances between the nodes on the edges. The distance between any two points can be found by summing up the lengths of the edges that belong to the shortest path between the two points.

The graph used in this paper is constructed using Berlin's spatial datasets from [1], structured in separate CSV files for the crossroads, roads and points of interest. For the implementation of the operators the datasets are imported into a graph structure of nodes and edges, where each node has a unique id, its latitude and longitude and a list of PoIs that have been mapped to it and each edge has a source and destination node and the distance between the two nodes in kilometers as parameters. Each PoI is mapped to the nearest crossroad and has a unique id, a type, its latitude and longitude and the distance to the node it is mapped to.

The map used for implementation and testing is the road network of Berlin, with 428769 crossroads, 504229 roads, 5548 PoIs and 7 category types: restaurant (2081), coffee shop (1002), atms/banks (597), movie theaters (141), pharmacies (589), pubs/bars (958), gas stations (180).

1.3 Challenges

Todo: Challenges

1.4 Contributions

Todo: Contributions

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

Notations and Preliminaries

In this chapter, I would like to introduce some terms, notations and definitions that are used throughout the thesis, such as the definition for a sequenced route query (SRQ), which we need in order to define the operators.

PoIs sets: We assume that we have n sets U_1, U_2, ..., U_n, which contain points in a 2-dimensional space $\mathbb{R}^2$ and $dist(.,.)$ is a distance function, which obtains the distance between two points in a two dimensional road network. The sets U_i represent the data sets for the different categories of points of interest, e.g. restaurants, gas stations etc..

Category sequence: $M = (c_1, c_2, ..., c_l)$ is a sequence of categories, if $1 \leq M_i \leq n$ for $1 \leq i \leq l$, where n is the number of points sets U_i. The user is only allowed to ask for existing location types.

Route: $R = (r_1, r_2, ..., r_r)$ is a route, if $r_i \in \mathbb{R}^2$ for each $1 \leq i \leq r$. R_sp is a route that starts from the starting point sp: $R_{sp} = (sp, r_1, r_2, ..., r_r)$

Route length: The length of a route $R = (r_1, r_2, ..., r_r)$ is defined as:

$$length(R) = \sum_{i=1}^{r-1} dist(P_i, P_{i+1}) \tag{2.1}$$

For $r = 1$ $length(R) = 0$.

Sequenced route: Let $M = (c_1, c_2, ..., c_l)$ be a sequence of point of interest categories. $R = (r_1, r_2, ..., r_l)$ is a sequenced route that follows the category sequence M, if $P_i \in U_{M_i}$ where $1 \leq i \leq l$. The points of interest in the route should belong to the corresponding category sets, defined in the category sequence.

Optimal sequenced route (OSR) query: Given a sequence of categories $M = (c_1, c_2, ..., c_l)$ and a starting point sp in $\mathbb{R}^2$, $Q(sp, M)$ is the Optimal Sequenced Route (OSR) Query, which searches for the shortest (in terms of function $length$) sequenced route R that follows M.

$$length(sp, R) = dist(sp, P_1) + length(R) \tag{2.2}$$

All other sequenced routes that follow M are referred to as candidate sequenced routes (SR).

Table 2.1 summarizes all used notations.

Symbol	*Meaning*
U_i	a point set for a category in $\mathbb{R}^2$
$\lvert U_i\rvert$	cardinality of the set U_i
n	number of point sets U_i
$dist(.,.)$	distance function in $\mathbb{R}^2$
M	category sequence, $= (c_1, c_2, ..., c_l)$
$\lvert M\rvert$	l, size of sequence M = number of items in M
c_i	ith member of M
R	route, $= (r_1, r_2, ..., r_r)$
$\lvert R\rvert$	r, size of route R = number of points in R
r_i	ith point in R
$length(R)$	length of R
$length(sp, R)$	length of $R_{sp} = (sp, r_1, r_2, ..., r_r), = length(R_{sp})$
$Q(sp, M)$	sequenced route query

Table 2.1: Notations

Chapter 3

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 endpoint. 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 4

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

4.1 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.

4.1.1 Problem definition

The equality operator is defined as follows:

Equality operator: Given a sequence of categories $M = (c_1, c_2, ..., c_l)$, a starting point sp in $\mathbb{R}^2$ and indices i and j, where $r_i \in U_{M_i}$, $r_j \in U_{M_j}$ and $M_i = M_j$, $EQUAL(i, j)$ is an equality operator, which states that r_i and r_j in the found route $R = (r_1, r_2, ..., r_l)$ should be the same points of interest. $Q(sp, M, EQUAL(i, j))$ is a Sequenced Route (SR) Query, which searches for the shortest (in terms of function $length$) sequenced route R that follows M and where $r_i = r_j$.

4.1.2 Precomputations

In order to faster calculate the heuristic for the partial routes, the nearest neighbors of all PoIs' categories to each node are precalculated and kept in a 2-dimensional table in memory for easy access. For precalculation a modified Dijkstra is executed for every node, which terminates as soon as it reaches the nearest neighbors of every category to a the given graph vertex.

4.1.3 Proposed approach

The equality operator is designed using the PNE approach, proposed in [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.

Heuristic

For generating the routes and deciding which of them are worth further expanding on, the proposed approach uses an initially calculated *upper bound* of an artificially build OSR, which satisfies the equality condition, and compares it to a lower bound of a route, considered by the algorithm. The *lower bound* of a certain route represents the sum of its length and its heuristic. The heuristic of a certain PSR is the maximum distance out of the distances to PoIs from the set of categories that are yet to be expanded. 9

Algorithm

The algorithm for the equality operator as shown in 1 is constructed using multiple procedures.

In $heap$ the PSR which are to be examined by the algorithm are stored. It is sorted by lower bound of the PSR. $found$ stores a candidate SR. UB is the length of the candidate route and it is updated each time a full SR is found The update is performed in $trim$ 10

First, an optimal sequenced route is found using the PNE algorithm 8. It is checked, if the two PoIs that the user has asked to be equal, are equal in the OSR. If so, the OSR is returned, else the equality operator continues with the creation of an artificial SR $dummySR$ and the modified PNE algorithm.
Second, we artificially create a sequenced route from the optimal route, found by PNE, as seen in 2. The optimal route is changed, so that r_j is made to be equal to r_i and the length of the artificially created PSR is the initial upper bound, by which later partial sequenced routes are either kept or discarded.
The modified PNE algorithm 3 begins iterating all r_1 from the category set U_{M_1}, which are subsequent to sp in the and it compares the *lower bound*, generated by them, to the global *upper bound*. They are only considered in further steps of the algorithm, if the partial sequenced route has a smaller lower bound than the upper bound.
Next, the modified PNE algorithm acts as a PNE algorithm and it fetches partial sequenced routes from the heap and generates new routes. This process is repeated until the $heap$ runs out of PSR, which means that all possible candidate routes have been expanded and taken into account.
At each subsequent iteration of the algorithm 3 (line 9 to 26) there are four distinct cases depending on the length of the route. Case one 4 and four 7 follow the original PNE. Case two 5 is focused on finding the travel distance between r_{j-1} and r_i. In each of the cases after fetching the route first the *lower bound* of the fetched PSR is compared to the global *upper bound* to see if the route should be modified or discarded (lines 1 to 3). After that the PSR is modified accordingly and again a length check is performed before finally putting the route on the $heap$. The length check is performed to make sure that the PSR is not longer than the already found SR with a length, which is the upper bound UB.

In case one 4 we find PSR before r_j. Two modifications of the PSR are performed, which follow the PNE algorithm 8. In a) (line 4) the nearest neighbor to the last PoI in the PSR r_k in $U_{M_{k+1}}$ is found and the PSR is updated to contain r_{k+1} and placed back on the heap. In b) (line 11) the k-th nearest neighbor to the second to last PoI r_{k+1} in U_{M_k} is found and the last PoIs in the PSR r_k is replaced with it.

In case two 5 r_j is to be found. In a) (line 4) instead of finding the nearest neighbor like the PNE algorithm does, the travel distance between the last PoI in the PSR r_{j-1} and r_i is found, because we want r_j to be equal to r_i ind the route. In b) (line 12) the k-th nearest neighbor to the second to last PoI r_{j-2} in $U_{M_{j-1}}$ is found and the last PoIs in the PSR r_{j-1} is replaced with it.

In case three 6 r_{j+1} is to be found. In a) (line 4) the nearest neighbor to the last PoI in the PSR r_j in $U_{M_{j+1}}$ is found and the PSR is updated to contain r_{j+1} and placed back on the heap. In b) (line 12) the k-th nearest neighbor to the second to last PoI r_{j-1} in $U_{M_{j-2}}$ is expected to be found, but in our case this is r_j and we have already calculated the travel distance between r_{j-1} and r_i in case two 5, so here nothing further needs to be done.

In case four 7 we find PSR after r_j. The case is similar to case one 4, except that in a) instead of directly putting the PSR on the heap, trimming 10 is performed to check if the route is a full SR and if the candidate route $found$ and the upper bound UB must be updated (line 9).

Algorithm 1: equalityOperator

Input : $Q(sp, M = (c_1, c_2, ..., c_l)), EQUAL(i, j)$
Output: $R = (r_1, r_2, ..., r_l)$

1 $initialize\ heap$; `// Heap with PSR`
2 $initialize\ found$; `// The candidate SR`
3 $initialize\ UB$;
4 $optimalRoute$ =`PNE(`Q`);`
5 **if** $optimalRoute[i] = optimalRoute[j]$ **then**
6   optimal route has been found;
7   **return** $optimalRoute$;
8 **else**
9   `dummySR();`
10   `modifiedPNE();`
11 **end**

Correctness

Todo: Correctness

Procedure dummySR($optimalRoute$)

```
// Creating a dummy SR (partial sequence route)
   from the found optimal route; replacing r_j
   with r_i
dummySR = (r_1, r_2, ..., r_{i-1}, r_i, ..., r_i) ;  // First part of the
   route
dummySR ← add PNE(r_i, (c_{j+1}, ..., c_l)) ;
UB = length(dummySR) ;
place dummySR on the heap;
```

Algorithm 2: `modifiedPNE` returns Route

```
foreach r_1 in U_{M_1} do Checking the upper bound for every r_1 neighbor of
  sp in the category set U_{M_1}
    build a new PSR with r_1;
    LB = length(PSR) + heuristic(PSR);
    if LB <= UB then
        place the new PSR (r_1) on the heap;
    end
end
while heap is not empty do
    current = fetch a PSR from the heap;
    switch s = size(current) do
        case s <= j − 1 do Finding PSRs before r_j
            case1();
        end
        case s = j do Finding PSR containing r_j
            case2();
        end
        case s = j + 1 do Finding PSR after/containing r_j
            case3();
        end
        case s >= j + 2 do Finding PSRs after r_j
            case4();
        end
    end
end
return found
```

Procedure caseOne

```
LB = length(current) + heuristic(current);
// Heuristic check
if LB <= UB then
    a) nearestNeigbour(r_k, U_{M_{k+1}});
    update PSR to contain r_{k+1};
    // Length check
    if length(PSR) <= UB then
        place PSR on the heap;
    end
end
b) kNearestNeigbour(r_{k-1}, U_{M_k});
update PSR;
if length(PSR) <= UB then
    place PSR on the heap;
end
```

Procedure caseTwo

```
LB = length(current) + heuristic(current);
// Heuristic check
if LB <= UB then
    a) dist(r_{j-1}, r_i);
    update PSR to contain r_i in the place j;
    // Length check
    if length(PSR) <= UB then
        // Trimming part
        trim(PSR);
    end
end
b) kNearestNeigbour(r_{j-2}, U_{M_{j-1}});
update PSR;
if length(PSR) <= UB then
    place PSR on the heap;
end
```

Procedure caseThree

```
LB = length(current) + heuristic(current);
// Heuristic check
if LB < UB then
    a) nearestNeigbour(r_j, U_{M_{j+1}});
    update PSR to contain r_{j+1};
    // Length check
    if length(PSR) <= UB then
        // Trimming part
        trim(PSR);
    end
end
b) // Found in caseTwo
```

Procedure caseFour

```
// Same procedure as caseOne() + trimming part to
   filter SR and update UB if needed
LB = length(current) + heuristic(current);
// Heuristic check
if LB <= UB then
    a) nearestNeigbour(r_k, U_{M_{k+1}});
    update PSR to contain r_{k+1};
    // Length check
    if length(PSR) <= UB then
        // Trimming part
        trim(PSR);
    end
end
b) kNearestNeigbour(r_{k-1}, U_{M_k});
update PSR;
if length(PSR) <= UB then
    place PSR on the heap;
end
```

Algorithm 3: `PNE` [5]

```
// Incrementally create the set of candidate routes for Q(sp, M) from starting point sp towards PoI set U_{M_l}
// 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 $s = size(PSR)$ **do**
- **case** $s == l$ **do**
  - PSR is the optimal route;
- **end**
- **case** $l \neq m$ **do**
  - a) `nearestNeigbour(`$r_{|PSR|}, U_{M_{|PSR|+1}}$`)`;
  - update PSR and put it back on the $heap$;
  - b) `kNearestNeigbour(`$r_{|PSR|-1}, U_{M_{|PSR|}}$`)`;
  - generate a new PSR and place it on the $heap$;
- **end**

end

Procedure heuristic(R)

```
// Calculates the heuristic for the given route R = (r_1, r_2, ..., r_k)
// For every route, which already contains r_i R = (r_1, r_2, ..., r_i, ..., r_k) the distance to r_j is calculated as the dist(r_k, r_i)
```

for c_{k+1} *to* c_n **do** For all direct neighbors to r_k of every subsequent category find the maximum distance
- find maximum;

end

Procedure trim(PSR)

```
if size(PSR) = m then
    if length(PSR) <= UB then
        update UB;
        update found;
    end
else
    place PSR on the heap;
end
```

4.1.4 Baseline approach

The baseline approach on the equality operator is entirely based on PNE by simply forcing r_i and r_j to be equal in the process of modifying the routes. In this variant, there is no heuristic and also no length checks.

Algorithm

The algorithm (shown in 11) starts by finding an optimal sequenced route with PNE, as mentioned in the proposed approach 4.1.3 and checks if r_i and r_j are already equal. If this is the case, it returns the found optimal route, otherwise it continues with the modified PNE 12.

Modified PNE proceeds with examining the routes on the *heap*, ordered by length, by size and modifying them according to PNE. When the current route on the heap is a full SR, then the optimal route has been found. The four cases (line 5, 13, 21 and 27) correspond to the cases in algortihm of the proposed approach, with the only difference being that no heuristic and length checks are performed.

Correctness

Todo: Correctness of baseline

4.1.5 Experimental study

Todo: Experiments

Algorithm 4: equalityOperator-baseline

Input : $Q(sp, M = (c_1, c_2, ..., c_l)), EQUAL(i, j)$
Output: $R = (r_1, r_2, ..., r_l)$

```
initialize heap ;                                // Heap with PSR
optimalRoute = PNE(Q) ;
if optimalRoute[i] = optimalRoute[j] then
    optimal route has been found;
    return optimalRoute;
else
    modifiedPNE-baseline;
end
```

Algorithm 5: `modifiedPNE`-baseline returns Route

$firstPSR$ = `nearestNeigbour(`sp, U_{M_1}`);`
place $firstPSR$ on heap;
$current$ = fetch a PSR from the $heap$;
switch $s = size(current)$ **do**
  case $s <= j - 1$ **do** Finding PSRs before r_j
    a) `nearestNeigbour(`$r_k, U_{M_{k+1}}$`);`
    update PSR to contain r_{k+1};
    place PSR on the $heap$;
    b) `kNearestNeigbour(`r_{k-1}, U_{M_k}`);`
    update PSR;
    place PSR on the $heap$;
  end
  case $s = j$ **do** Finding PSR containing r_j
    a) `dist(`r_{j-1}, r_i`);`
    update PSR to contain r_i in the place j;
    `trim(`PSR`);` `// Trimming part`
    b) `kNearestNeigbour(`$r_{j-2}, U_{M_{j-1}}$`);`
    update PSR;
    place PSR on the $heap$;
  end
  case $s = j + 1$ **do** Finding PSR after/containing r_j
    a) `nearestNeigbour(`$r_j, U_{M_{j+1}}$`);`
    update PSR to contain r_{j+1};
    `trim(`PSR`);` `// Trimming part`
    b) `// Found in caseTwo`
  end
  case $s >= j + 2$ **do** Finding PSRs after r_j
    a) `nearestNeigbour(`$r_k, U_{M_{k+1}}$`);`
    update PSR to contain r_{k+1};
    `trim(`PSR`);` `// Trimming part`
    b) `kNearestNeigbour(`r_{k-1}, U_{M_k}`);`
    update PSR;
    place PSR on the $heap$;
  end
  case $s == l$ **do** Optimal route with equal PoIs at i and j has been found
    return $current$;
  end
end
return $found$;

4.2 Not-Equality operator

The equality operator is based on the need to express that some PoIs in the SRQ of the same category shouldn't be equal.

4.2.1 Problem definition

The not-equality operator is defined as follows:

Not-Equality operator: Given a sequence of categories $M = (c_1, c_2, ..., c_l)$, a starting point sp in $\mathbb{R}^2$ and indices i and j, , where $r_i \in U_{M_i}$, $r_j \in U_{M_j}$ and $M_i \neq M_j$, $NOTEQUAL(i, j)$ is an equality operator, which states that r_i and r_j in the found route $R = (r_1, r_2, ..., r_l)$ should be different points of interest. $Q(sp, M, NOTEQUAL(i, j))$ is a Sequenced Route (SR) Query, which searches for the shortest (in terms of function $length$) sequenced route R that follows M and where $r_i \neq r_j$.

4.2.2 Precomputations

The precomputations are the same as for the equality operator in Section 4.1.2.

4.2.3 Proposed approach

The not-equality operator is designed using the PNE approach, proposed in [5]. It uses the progressive neighbour explorator as its base to upgrade on and explore all the possible optimal routes until it finds an optimal route, in which the given PoIs are different.

Algorithm

The algorithm (shown in 13) starts by initializing the heap, ordered by the length of the routes, and the first PSR (line 1, 2, 3). It then proceeds to inspect and modify the routes on the heap based on their length, until the $heap$ is empty. Each full SR (line 7 to 13) is checked if it satisfies the requirement for r_i and r_j to not be equal. If this is the case, the found optimal SR is returned, otherwise the next PSR on the heap is fetched. If the fetched PSR is not a full SR but a partial route (line 14 to 19), then the algorithm performs a) and b) as in the PNE algorithm 8.
Modified PNE proceeds with examining the routes on the heap by size and modifying them according to PNE. When the current route on the heap is a full SR, then

the optimal route has been found. The four cases (line 5, 13, 21 and 27) correspond to the cases in algorithm of the proposed approach, with the only difference being that no heuristic and length checks are performed.

Correctness

Todo: Correctness

4.2.4 Experimental study

Todo: Experiments

Algorithm 6: notEqualityOperator

Input : $(sp, M = (c_1, c_2, ..., c_l)), NOTEQUAL(i, j)$
Output: $R = (r_1, r_2, ..., r_l)$

$initialize\ heap$;
$firstPSR$ =`nearestNeigbour(`sp, U_{M_1}`);`
place $firstPSR$ on heap;
`// At each iteration of PNE a` PSR `(partial sequenced route) is fetched and examined based on its length and it is checked`
fetch a PSR from the $heap$;
while *heap is not empty* **do**
    if $length(PSR) = l$ **then**
        if $PSR[i] \neq PSR[j]$ **then**
            PSR is the optimal route;
            return PSR;
        else
            `// We continue with the next PSR on the heap`
        end
    else
        a) `nearestNeigbour(`$r_{|}PSR|, U_{m_{|PSR|+1}}$`);`
        update PSR and put it back on the $heap$;
        b) `kNearestNeigbour(`$r_{|PSR|-1}, U_{m_{|PSR|}}$`);`
        generate a new PSR and place it on the $heap$;
    end
end
; `// In the rare case that there is only one PoI of the given category, the algorithm would return an empty route`
return $null$;

Chapter 5

Evaluation

The completeness of the operators stems from

Chapter 6

Conclusion and Future work

Bibliography

[1] E. Ahmadi and M.A. Nascimento. Datasets of roads, public transportation and points-of-interest in Amsterdam, Berlin and Oslo. In: `https://sites.google.com/ualberta.ca/nascimentodatasets/`, 2017.

[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.