Shortest Path Cache

June 14, 2012

Jeppe Rishede Thomsen
Department of Computing
Hong Kong Polytechnic University
Kowloon, Hong Kong
csjrthomsen@comp.polyu.edu.hk

Man Lung Yiu
Department of Computing
Hong Kong Polytechnic University
Kowloon, Hong Kong
csmlyiu@comp.polyu.edu.hk

Symbol	Meaning
$QR_{q,R}$	Range query from q to all $o_i : d_{q,o_i} < R$)
$\chi_{s,t}$	The frequency of a SP
$P_{s,t}$	A SP from s to t
$d_{s,t}$	The SP distance of a path $P_{s,t}$
$euclid_{s,t}$	The euclidean distance of a path $P_{s,t}$
$\mathcal{O}$	The set of POI $\in \mathbf{V}$, the set of vertices
o_i	Element i in $\mathbf{O}$
$\mathcal{O}_{q,R}$	The result set of $Q_{q,R}$
$dist_{\mathcal{O}}$	Table of distances between vertices $v_s, v_t \in P_{s,t}$: $P_{s,t} \in$ SP$_q$
SP$_q$	A set of SP from q to all o_i
Ψ	The Cache
$G(\mathbf{V}, \mathbf{E})$	Graph representation of the Map

Table 1: Table of Symbols

ABSTRACT

abstract...
Point of Interest (POI)
Shortest Path (SP)

1. INTRODUCTION

$$\chi_{s,t} = |P_{s,t}| \in ((q, R), \mathcal{O}) \quad (1)$$

Definition Range Search
A range search query, denoted by $Q_{q,R}$ consist of a source vertice q and a range $\mathbf{R}$. The result of $Q_{q,R}$, denoted $\mathcal{O}_{q,R}$, is a set of objects $o_i \in \mathcal{O}$ with SP distance $d_{s,t} \leq R$ on graph $G(\mathbf{V}, \mathbf{E})$.

Definition Range Search Query Log
A range search query log $\mathcal{RQL}$ is a collection of timestamped queries that have been issued by users in the past. A query is on the form (q, R), where q is a query point and R is a range. The full form of the log, $\mathcal{RQL}$, is then: $\{(q_0, R_0), \ldots, (q_i, R_i)\}$

Proceedings of the VLDB Endowment, Vol. 5, No. 0

Algorithm 1: Straightforward naive Algorithm

input :
(q, R): A Range query
$\mathcal{O}$: A set of POI
output:
A set POI $\in \mathcal{O}$

```
1 Naive((q, R), O)
2     tmpResult ← Check Ψ for any P_{q,o_i}
3     foreach o_i ∈ O do
4         if P_{q,o_i} ∉ tmpResult then
5             tmpResult ← Calculate P_{q,o_i}  //SP from q to o_i
6     foreach P_{q,o_i} ∈ tmpResult do
7         if d_{q,o_i} < R then
8             result ← P_{q,o_i}
9     return result
```

Algorithm 2: Fair Algorithm

input :
(q, R): A Range query
$\mathcal{O}$: A set of POI
output:
A set POI $\in \mathcal{O}$

```
1 Fair((q, R), O)
2     foreach o_i ∈ O with euclidean distance < R from q do
3         pruned_O ← o_i
4     result ← Naive((q,R), pruned_O)
5     return result
```

Using a query log, $\mathcal{RQL} : \{(q_0, R_0), \ldots, (q_i, R_i)\}$, we first need to expand each query into its SP result set containing SPs from q to each POI reachable within R distance on a roadnetwork. To find the result set of a query we use one of two algorithms: **Naive** or **Fair**.

The **Naive** algorithm will not only find the result set of SP for each query $\in QL$, but also the SP from q to all other possible POI, before pruning and returning the result set of POI's. **Naive** does so by first finding all paths from q to any POI in the cache. Afterwards it will issue SP queries from q to the remaining POI's not already found via the cache. Once all paths are found **Naive** will prune the set of paths such that only the paths with distance < R remain. This is the result so **Naive** returns it to the user.

The idea behind the **Fair** algorithm (see alg. 2) is to prune POI's, which can not belong to the result set of a query, before doing range

Algorithm 3: Range search Algorithm

input : A touple $((q, R), \text{SP}_q, dist_O)$
(q, R): Range query
SP_q: A set of SP from q to all o_i
$dist_O$: table of distances between vertices
output: A set POI $\in \mathcal{O}$

1 ***RSearch***$(((q, R), \text{SP}_q, dist_O))$
2 using SP_q and $dist_O$ find all P_{q,o_i} such that d_{q,o_i} is smaller than R.
3 **return** $Q_{q,R}$

search. The **Fair** algorithm works by, for a query (q,R), first finding the set *tmpPOI*, of POI's which lay within euclidean distance R of the query point q. **Fair** then invokes the **Naive** algorithm using

first finding a MBR A of the region defined by R on the road-network, it then extracts all POI within A and invokes **Naive** using this limited set of POI.

The reasoning behind finding the SP to more than just the answer of a query is because when we later check the cache for an answer we have no way of knowing when we have extracted all the SP belonging to the result set of a query, so we need to store at least some extra points laying outside R, so that we know that once we have checked those do not belong in the result, then no other POI can be missing from the result set.

An example, using figure 1[1], would be that for query point $q_1 - q_3$, **Naive** will find the SP from q to all POI (white points), while **Fair** will only return the exact result for q_1. For q_2 it will find the result plus the SP for one extra POI. For q_2 it finds the SP for two extra POI.

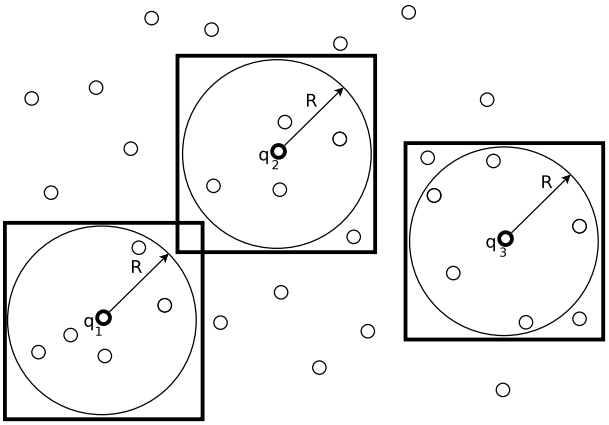

Figure 1: Query points q_1, q_2, q_3, with region defined by R. White points are POI

1.1 Better pruning via dominating set

As we can see in figure 1, then using an MBR to prune may not provide the best result. The problem is that we need to return enough extra SP to POI outside R so that we are able to check if we have the complete result when a similar query is issued again (assuming that all the paths have been added to the cache after the initial query).

[1]R is defined as the shortest path distance, so the region will almost never be a circle, but it is show as such for simplicity, the roadnetwork is also ignored in the figure

In figure 1 we see that for q_1 no additional points outside the answer set is retained, if we only calculate and add the SP of these POI to the cache, then we have no way of checking if we have the full result in the cache next time a similar query is issued, except invoking the SP API. Invoking the SP API is what our cache is trying to avoid.

Figure 2 shows a query where the all the white POI outside are closer to q on at least on dimension, than any black point in the figure. If we add the set of white POI to the cache then we will be sure that we can answer a similar query without invoking the SP API.

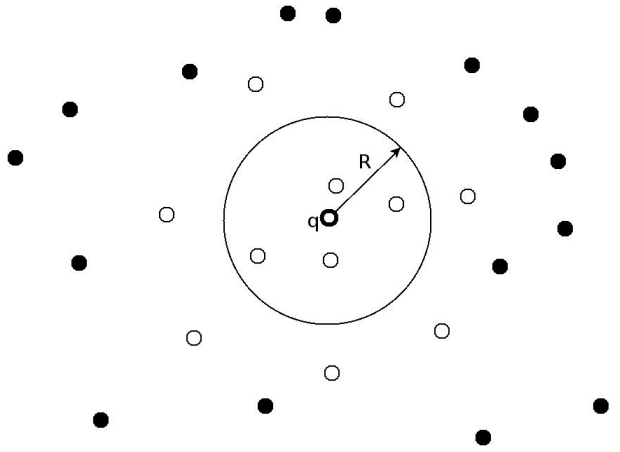

Figure 2: White points outside region R are dominating all black points