Shortest Path Cache

September 19, 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 $o_i : d_{q,o_i} \leq R$)
$Q_{s,t}$	SP query from s to t
$\chi_{s,t}$	The frequency of a SP
$P_{s,t}$	A SP from s to t
$d_{s,t}$	The SP distance of a path $P_{s,t}$
$\mathfrak{d}_{s,t}$	$\|t - s\|_2$, the euclidean distance
$\mathcal{O}$	The set of $POI \in \mathbf{V}$, the set of vertices
o_i	Element i in $\mathcal{O}$
$\mathcal{O}_{q,R}$	The result set of $o_i \in Q_{q,R}$
$dist_{\mathcal{O}}$	Table of distances between vertexes $v_s, v_t \in P_{s,t} : P_{s,t} \in SP_q$
SP_q	A set of SP from q to all o_i
$\mathcal{QL}_{\mathcal{R}}$	Query log of range search queries
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

$$\chi_{s,t} = |\{t : QR_{s,R} \in \mathcal{QL}_{\mathcal{R}}, t \in \mathcal{O}, R \in \mathbb{R}, t \in P_{s,t}\}| \quad (1)$$

$$\chi_{s,t} = |\{t : QR_{s,R} \in \mathcal{QL}_{\mathcal{R}}, \{t \in \mathcal{O} | \mathfrak{d}_{s,t} \leq R\}, R \in \mathbb{R}, \exists P_{s,t} | d_{s,t} \leq R\}| \quad (2)$$

Definition Range Search
A range search query, denoted by $Q_{q,R}$ consist of a source vertex 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 $G(\mathbf{V}, \mathbf{E})$.

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

Algorithm 1: Naive Algorithm

input :
(q, R): A Range query
$\mathcal{O}$: A set of POI
output:
A set POI $\in \mathcal{O}$
Naive$((q, R), \mathcal{O})$
  foreach $o_i \in \mathcal{O}$ **do**
    if $P_{q,o_i} \in \Psi$ **then**
      insert o_i into $result$
    Call SP API on Q_{q,o_i} *//SP from q to* o_i
    if $d_{q,o_i} \leq R$ **then**
      insert o_i into $result$
  return *result*

Algorithm 2: Fair Algorithm

input :
(q, R): A Range query
$\mathcal{O}$: A set of POI
output:
A set POI $\in \mathcal{O}$
Fair$((q, R), \mathcal{O})$
  foreach $o_i \in \mathcal{O} : \mathfrak{d}_{q,o_i} \leq R$ **do**
    $candidate_{\mathcal{O}} \leftarrow o_i$
  result $\leftarrow$ **Naive**((q,R), $candidate_{\mathcal{O}}$)
  return *result*

Definition Range Search Query Log ($\mathcal{QL}_{\mathcal{R}}$)
A range search query log $\mathcal{QL}_{\mathcal{R}}$ 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, $\mathcal{QL}_{\mathcal{R}}$, is then: $\{(q_0, R_0), \ldots, (q_i, R_i)\}$

Using a query log, $\mathcal{QL}_{\mathcal{R}} : \{(q_0, R_0), \ldots, (q_i, R_i)\}$, 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 road network. 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 $\in \mathcal{QL}_{\mathcal{R}}$, 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 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 *tmpPOI* as the set of POI. The pruning method of **Fair** is correct as the euclidean distance R is also the longest SP, with length R, possible. It is however important to note that this pruning technique only work if the weights of edges in a road network are distances.

Equation 1 and 2 define the frequency of a SP in Ψ, using **Naive** or **Fair** respectively.

An Example of how **Fair** might work is shown in fig. 1. In it we have 2 queries: q_1 and q_2. Each query have a range $R = 13$ associated with it, which is used to form the pruning areas (circles) of the queries. The pruning areas are the shown by the 2 circles, each covering a query. The white circles are all vertices. $v_1, \ldots, v_{11}$ are the regular vertices. $o_1, \ldots, o_{10}$ represent the POI's, and q_1, q_2 are the query points. Vertices ($\mathbf{V}$), and connecting lines,edges ($\mathbf{E}$), model a road network, $G(\mathbf{V}, \mathbf{E})$.

In our problem the range R is constrained on a road network, meaning that after pruning, when the vertexes more than R euclidean distance away are no longer considered, the remaining vertexes may still be further than R distance in the road network. We can see an example of this with q_2, where only one out of the 3 POI in the gray region is actually within R SP distance (o_9, the reader should visually easily be able to confirm this using the edge weights in fig 1).

When we consider what to put into the cache, then we will include SPs for all POI within the pruning area, regardless of whether the POI's are contained in the result set or not. The reasoning behind this, is that if we only add the exact result, then we will not be able to determine if the entire result is in the cache or not, except in the special case that all vertexes in the pruning region is part of the result, as with query q_1.

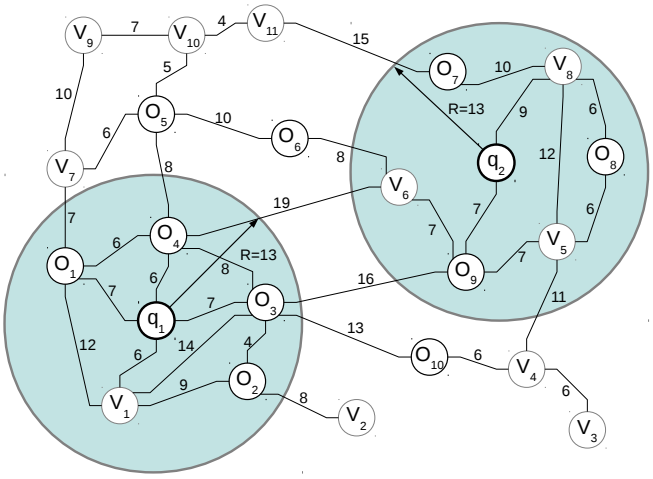


Figure 1: Query points q_1, q_2, q_3, with region defined by R. White points are POI

2. EXPERIMENTS

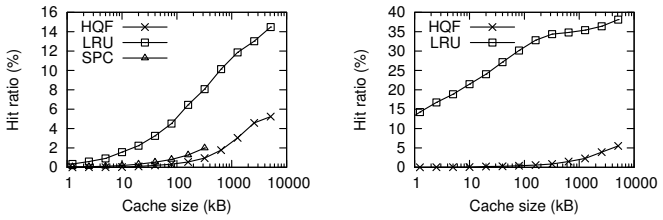
2.1 Query Sources & Generation

2.2 Experimental Setting

Our 3 caching methods, SPC, HQF, and LRU, share a number of common settings which, unless stated otherwise, will be set to their default values: The number of levels in the kD-tree are 14 (i.e. 16.384 regions). We will use the list cache representation as the default cache representation, where each vertex use one byte. The default cache size is set to 5.120.000 bytes.

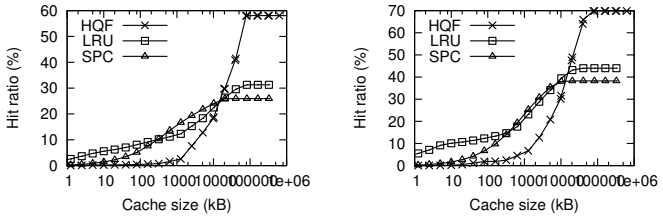
2.3 Proxy

2.3.1 Range



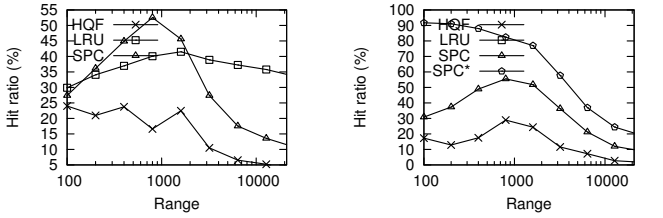
(a) Aalborg (b) Beijing

Figure 2: Hit ratio vs. cache size, NAIVE algorithm



(a) Aalborg (b) Beijing

Figure 3: Hit ratio vs. cache size, FAIR algorithm



(a) Aalborg (b) Beijing

Figure 4: Hit ratio vs. Range, FAIR algorithm

2.4 Server