Information Systems 49 (2015) 142–162

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

On using broadcast index for efficient execution of shortest path continuous queries

Chun Jiang Zhu [a], Kam-Yiu Lam [a,*], Reynold C.K. Cheng [b], Chung Keung Poon [c]

[a] Department of Computer Science, City University of Hong Kong, Hong Kong
[b] Department of Computer Science, University of Hong Kong, Hong Kong
[c] School of Computing and Information Sciences, Caritas Institute of Higher Education, Hong Kong

ARTICLE INFO

Article history:
Received 5 February 2014
Received in revised form
9 December 2014
Accepted 10 December 2014
Recommended by: F. Korn
Available online 18 December 2014

Keywords:
Wireless data broadcast
Broadcast index
Road networks
Continuous queries and shortest path searching

ABSTRACT

Various efficient *indexing techniques* have been proposed for formulating broadcast index to minimize the tune-in cost at mobile clients to support shortest path search in a road network. However, none of them is designed for *shortest path continuous queries* (*SPCQ*). Due to frequent updates of traffic data, an *SPCQ* has to be invoked periodically while the client is following the suggested shortest path to its destination. This could result in heavy path searching cost at the client. To reduce the tune-in and path searching costs at the clients, in this paper, we propose the *progressive approach* (*PA*) in which we apply the *next region* (*NR*) approach together with a hierarchical network approach to formulate the local networks at the clients for shortest path navigation. Since the shortest path may change due to changing traffic, to minimize the path searching cost, we aim for an approximate path including shortcuts of *shortcut networks* (*SN*) from each invocation of an *SPCQ* to navigate the client to its destination *step-by-step*. In formulating the client local network, higher level shortcut networks are chosen if the regions are farther away from the current region of the client. An important property of the approximation mechanism adopted in *PA* is that traffic updates may be pruned if they will not significantly affect the shortest path information to be distributed to the clients. This can reduce the index re-generation cost at the traffic server. Extensive performance evaluation experiments have been conducted to investigate how *PA* reduces the tune-in cost from data broadcast and path searching cost at the clients with just small delays in the arrival times to their destinations.

1. Introduction

With the latest advances in mobile communication and pervasive computing technologies, the design of higher performance *location-based services* has received tremendous interests in recent years, e.g., Google map services and searching nearby restaurants [7,5]. Various types of spatial queries have been intensively studied and efficient algorithms have been proposed for processing spatial queries [6,4,10,21]. One of the important types of location-based services is the shortest path search in road navigation [25,23,26]. In this paper, we study the *shortest path searching problem* in a *road network* where queries are submitted from mobile clients in vehicles to inquire the *shortest paths* from current locations to their specific destinations. We consider a road network which maintains complex road connections of a big city, e.g., Hong Kong. The roads in these cities contain a lot of traffic lights, junctions, small single lane roads, and roadside parking and stores. In such a road network, it is difficult to predict the traffic conditions of the roads and traffic jam could happen suddenly. Therefore,

* Corresponding author.
E-mail addresses: chunjizhu2-c@my.cityu.edu.hk (C.J. Zhu), cskylam@cityu.edu.hk (K.-Y. Lam), ckcheng@cs.hku.hk (R.C.K. Cheng), ckpoon@cihe.edu.hk (C.K. Poon).

http://dx.doi.org/10.1016/j.is.2014.12.005
0306-4379/© 2015 Elsevier Ltd. All rights reserved.

a close monitoring on the traffic has to be performed in order to provide effective shortest path navigation.

In the road network, each road segment (a road may be divided into several segments according to its connections) is represented by a directed edge together with a weight to show the latest measured traffic condition of the road segment (i.e., the estimated travel time from one end of the road segment to the other end). It is assumed that the road network is managed by a *traffic server*. It receives measurements of traffic updates from road sensors to refresh the weights of the road segments to reflect their latest traffic conditions.

In processing a shortest path query, the shortest path to the specified destination is obtained by searching the road network to get the path with the *shortest expected travel time* after considering the *current traffic* of the road segments. Since the number of clients, which may submit shortest path queries, could be large, an efficient method for processing the shortest path queries is to use *data broadcast* [14,34] to *push* traffic data, which contains information about the connections of the road segments as well as their weights (i.e., their traffic conditions), to the clients to execute the queries locally.

To reduce the energy cost at the clients for getting traffic data, a common approach adopted in data broadcast is to prepare a *broadcast index* as the header of each broadcast cycle [20,30]. The broadcast index defines the broadcast times of traffic data in a broadcast cycle. How to prepare the broadcast index to minimize the amount of traffic data to be required by a client for processing its shortest path query is an important research problem and various efficient *indexing* techniques, e.g., elliptic boundary (*EB*) [17], next region (*NR*) [17] and contraction hierarchies (*CH*) [24] have been proposed. In these methods, the index contains the shortest path information prepared by the traffic server for the clients to obtain from the broadcast cycle according to their current locations and destinations.

Since the shortest path to a client's destination may change due to changes in road traffic, the shortest path query has to be re-executed periodically until the client has arrived its destination. For example, as shown in Fig. 1, the client obtains a shortest path to go from current location s to its destination d at time t (drawn in thick lines). However, when it is at location a at time $t+\delta$, after a number of updates to the road segments, to ensure the current path is still the shortest one (actually, the shortest path from b to c has changed to the dash lines), the shortest path search has to be re-executed. We call this kind of queries as the *shortest path continuous queries (SPCQ)* as they will be re-executed until a certain condition is satisfied, i.e., arriving the destinations or expiring their deadlines. A typical example of *SPCQ* is the ambulance services in which the arrivals of ambulances to their destinations have pre-defined deadlines and the changes in road traffic need to be closely monitored.

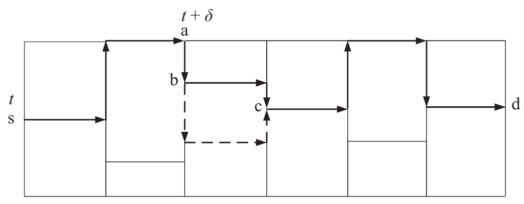


Fig. 1. An illustration of *SPCQ*.

Although various pre-processed indexing techniques, e.g., *NR*, have been shown to be effective in reducing the tune-in cost as well as path searching cost at mobile clients, they are not designed for *SPCQ* and may not be efficient in handling changing road traffic. A change in traffic data may change the shortest path for a client and the broadcast index may need to be re-generated after traffic updates. This could result in heavy index re-generation cost at the traffic server as well as heavy tune-in and re-searching costs at the clients. Another efficient technique to support shortest path searching costs is to use the *hierarchical indexing technique*, e.g., *HiTi* [16] and *LTI* [28]. In *HiTi* and *LTI*, a hierarchy of network index is built from the road network and different levels of the sub-networks (or called sub-graphs in [16,28]) are selected to construct a local road network for a client to search the shortest path to its destination. By merging lower level sub-graphs into higher level sub-graphs, the number of nodes in the network can be greatly reduced resulting in lower path searching cost.

In this paper, based on the road network model introduced in [17], in which the road network is divided into a set of connected and non-overlapping regions, we propose the *progressive approach (PA)* with the purposes of minimizing the tune-in and path searching costs at mobile clients. Note that the processing power of many client devices, e.g., smartphones, is limited, and at the same time other applications may be executing at the clients concurrently, e.g., receiving broadcast news. Although the clients may be able to obtain energy supply from car chargers, minimizing the total energy requirement is still highly preferable, since energy conservation is always a great concern in the design of future intelligent automobile systems, e.g., electric cars [8,27–29]. To reduce both the tune-in and path searching costs, similar to *NR*, in *PA*, the traffic server performs a pre-processing to calculate a set of required regions for each client to build its local network for path searching. To reduce the index re-generation cost at the traffic server, *update thresholds* are defined to remove some traffic updates that will not significantly affect the current shortest path information to be distributed to the clients. To further reduce the number of vertices in the local networks, similar to *HiTi* and *LTI*, in *PA*, the local network constructed by a client is a simplified hierarchical network, called *hierarchical shortcut network* to navigate the client to its destination. A more detailed network will be generated from each invocation of the *SPCQ* while the client is following the suggested shortest path to its destination until it reaches its destination or the deadline expires.

The remaining parts of the paper are organized as follows. In Section 2, we review the important indexing techniques for supporting the shortest path search at mobile clients using data broadcast. In Section 3, we introduce the system model assumed in this paper. The performance problem of *SPCQ* over a dynamic road network is illustrated with an example in Section 4. In Section 5, we introduce the principles of the *progressive approach (PA)* which consists of two main parts: generation of *shortcut network* (*SN*) and generation of *multi-level shortcut networks* (*MLSN*). In Section

6, we discuss how *SN* are formulated. The generation of *MLSN* and broadcast index, as well as the process logical of *SPCQ* at clients and cost analysis are presented in Section 7. In Section 8, the performance evaluation of *PA* is reported. We conclude the paper with a brief discussion on the future work in Section 9.

2. Related work

Data broadcast has been shown to be an efficient approach to distribute a large amount of data to a large group of mobile clients through a wireless network [1,14]. To improve the efficiency in data distribution and support the processing of different types of spatial queries, including the shortest path queries, a number of research works have been done on one-dimensional [13,14] and multi-dimension indexing [32,33,22]. For example, the Hilbert curve index (HCI) [32], designed for range queries and KNN queries, was adopted to convert the two-dimensional space into one-dimensional one, by using the Hilbert space-filling curve. Then the data can be treated as one-dimensional data and indexed by B^+-tree to be broadcast on the air. To reduce the waiting time for getting broadcast data from HCI, the distributed spatial index (DSI) was proposed by sacrificing the energy consumption issue [33]. DSI also adopts the Hilbert spacing-filling curve. The data ordered in Hilbert values are divided into *frames* with fixed number of data and each frame contains an index table.

To support snapshot queries, continuous range queries and KNN queries, the broadcast grid index (BGI) was introduced by Mouratidis et al. [22]. In BGI, a regular grid is imposed on the road network to generate a number of cells. The information (the ceil's extent, the number of data objects inside and their coordinates) of all the cells is organized in a certain implicit sequence for forming the index.

All the above methods are focused on spatial queries on Euclidean space. Kellaris and Mouratidis [17] proposed the *NR* (next region) and *EB* (elliptic boundary) broadcast indices for shortest path queries on road networks, in which no Euclidean space was assumed. Both *NR* and *EB* aim to minimize the energy cost at the clients for getting the required road data. They divide the road network into a set of non-overlapping regions and the server prepares sub-sets of regions, called *required regions*, in which the shortest path information can be obtained by mobile clients at different regions to their destinations. Empirically, *NR* was shown to be better than *EB* in terms of energy cost and response time in getting road data.

Jing et al. [15] proposed the broadcast index called *BagIndex* for processing of shortest path queries. *BagIndex* is based on *tree decomposition*, which is a tree with each tree node, termed as a *bag*, containing a group of vertices. This structure ensures that for any two vertices the distance is preserved by a tree path between two bags containing the two vertices. To search the shortest path, a client just needs to capture the corresponding path of bags in the tree decomposition. Experimentally, it was shown that in the best case its energy cost was four times lower than that of *NR* but with response time five times larger compared with *NR*.

All the above methods assumed that the road network is static, i.e., no updates to the edges. When there are insertion or deletion of edges and changes of edge weights, the *contraction hierarchy (CH)* [11,12] can deal with them and support shortest path searching over such a dynamic network. *CH* consists of two phases: a node ordering phase and a hierarchy construction phase. In order to maintain a correct structure of the network after updates, *CH* reuses the node orders and maintains the hierarchy by only updating the affected nodes. However, if the number of changes is large, the number of affected nodes may become very large. Even worse, the node orders may be out-of-date and a new node ordering procedure may be required after updates. In addition, although the searching time can be very fast because of the addition of shortcuts, *CH* may not be preferable in wireless data broadcast, since it requires all the original road network and shortcuts added to be captured by the client resulting in a large tune-in cost.

Batz et al. [3] extended *CH* to *TCH* (time-dependent contraction hierarchy) by carefully performing the *contraction* operations during the pre-processing, and adding a *shortcut* whenever the path passing through the contracted vertex is a minimum cost path for *some departure time*. To achieve more energy saving in getting broadcast data by mobile clients for shortest distance queries, Poon and Zhu [24] designed *CH* (contraction hierarchy) index by exploiting the special properties of road networks. Based on the enhanced *CH* index, the *CHBN* index was proposed to provide a trade-off between energy cost and response time in getting the required data by a client via a user-tunable parameter. It was shown to give faster response time in getting broadcast data by clients than *CH* while still being energy efficient.

Another approach to reduce the index generation cost at the server is to construct a hierarchical road network to reduce the number of vertices in the client local road networks. In [16], *HiTi* was proposed for shortest path searching with support to a dynamic road network. In *HiTi*, the road network is partitioned into a set of sub-graphs which are merged recursively to generate a *HiTi* hierarchy. The current traffic data of each edge in the sub-graphs are then computed and stored at the *HiTi* broadcast index. Each client listens to the broadcast channel to obtain the latest traffic data for building a local graph consisting of different levels of sub-graphs according to its current location and destination for searching the shortest path to its destination. In formulating a local graph, the lowest level sub-graphs are chosen for the current location of the client and its destination, while higher level sub-graphs are selected for those in between these two sub-graphs. Since the local graph contains higher level sub-graphs of the original network, its size could be much smaller than the original road network resulting in lower path searching cost. Although *HiTi* has shown excellent performance at the traffic server in terms of the index generation cost, the path searching cost could still be expensive especially for systems where the computation power of the mobile devices is limited. In this paper, we would like to design an efficient index generation algorithm that can minimize both the tune-in and path searching costs at the clients.

In a recent work [28], the *live traffic index* (*LTI*) was proposed for solving the shortest path searching problem on a dynamic road network at mobile clients. Similar to *HiTi*, *LTI* is a hierarchical index of a road network. In *LTI*, an effective graph partitioning algorithm was proposed to minimize the total size of the sub-graphs for formulating the local road networks at the clients and a combinatorial optimization was provided for reducing the search space for the shortest paths. With *LTI*, a client can perform shortest path computation by only retrieving a portion of the entire index that has updated edge values. It was shown that *LTI* gave improved performance as compared with both *CH* and *HiTi* in terms of lower tune-in time for getting traffic data at the mobile clients, and lower the maintenance cost of the network hierarchy at the traffic server.

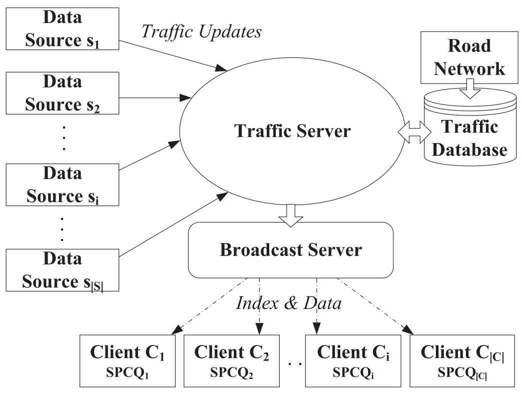


Fig. 2. The model of a road network system.

3. System model: the road network system

The system model of the road network system is shown in Fig. 2. It consists of six main components: a traffic server, a traffic database, a road network, a broadcast server, a set of mobile clients and a set of data sources. Table 1 summarizes the definitions of the set of symbols that are frequently used in the paper.

It is assumed that each directed edge e_i in the road network represents a road segment r_i in the service area (e.g., the physical road area covered by the road network) and is associated with a weight w_i to indicate the estimated travel time (or simply called "the travel time" in the rest of the paper) to complete the road segment r_i from one end to the other end following the direction of the road segment. The information (e.g., its location, length, connections, etc.) of each road segment r_i together with its weight w_i are recorded in a traffic data object d_i maintained in the traffic database at the traffic server. To simplify the discussion, it is assumed that there is only one powerful traffic server in the system.

Following the assumption in [17], the road network is partitioned into a set of connected and non-overlapping regions (by kd-tree partitioning) and each region consists of a group of connected road segments. To simplify the discussion, we assume that the regions are square in shape and have the same dimension represented by the edge length.

Table 1
Frequently used symbols and their definitions.

Symbol	Definition
e_i	The ith directed edge in the road network
r_i	The ith road segment represented by edge e_i
d_i	The traffic data object for road segment r_i
w_i	The estimated time to complete road segment r_i
s_i	The sensor that measures the traffic for road segment r_i
C_i	The ith mobile client
$SPCQ_i$	$SPCQ$ issued by client C_i
cp_i	Current location of client C_i
$dest_i$	Destination of $SPCQ_i$
$begin_i$	Begin time of $SPCQ_i$
end_i	End time of $SPCQ_i$
T_i	Time duration of $SPCQ_i$
R_j^i	The jth region at level i of region tree (R_j implies $i=1$, and R^i refer to region(s) at level i)
r^i	The number of regions/higher level regions at level i (r implies $i=1$)
S_i	The ith shortcut
$W(P)$	The total sum of travel time w_j for each road segment r_j on path P
$SN\ (R_j^i)$	The shortcut network of region R_j^i
$MLSN\ (R_i, R_j)$	The multi-level shortcut network at the traffic server for region pair (R_i, R_j)
$MLSN_C\ (R_i, R_j)$	The multi-level shortcut network at mobile clients for region pair (R_i, R_j)
$children\ (R_j^i)$	The set of R^{i-1} whose grouping resulting in R_j^i in a region tree
$parent\ (R_j^i)$	The R^{i+1} such that $R_j^i \in children(R^{i+1})$ in a region tree
$sibling\ (R_j^i)$	$children(parent(R^i)) \backslash R_j^i$ in a region tree
t_{thres}^i	The update threshold for level i
$Adj(G)$	The size of road network G in an adjacency list representation
$Dij(G)$	The average number of edges traversed during a Dijkstra search in G
$Bor(R^i)$	The number of border vertices in R^i
RN_C	Client road network for shortest path search at clients

A data source s_i measures the traffic condition for a road segment r_i periodically and generates traffic updates to be submitted to the traffic server for refreshing the value of the corresponding data object d_i, i.e., the new travel time w_i for r_i. Note that although installing traffic sensors on the roads is still not popular due to heavy installation and operation costs, the real-time traffic of a road can easily be estimated according to the movements of a group of vehicles, e.g., the bus and taxi systems, on the road. Since the real-time locations of vehicles can be obtained from their GPS devices, they can measure their travel times on the road segments in real-time and calculate the new traffic data for the road segments to be reported to the traffic server.

The clients are moving entities in the road network. A client C_i may submit a *shortest path continuous query*, $SPCQ_i$, to ask for going from its current location cp_i to its destination $dest_i$. Both cp_i and $dest_i$ are assumed to be defined in terms of one end of a road segment to simplify the discussion. It is assumed that each client is equipped with a positioning device, e.g., GPS, which can accurately determine its location at current time. Each $SPCQ_i$ is associated with a time duration (T_i) starting from its generation time and is formally defined as follows:

$$SPCQ_i(cp_i, dest_i, begin_i, end_i)$$

Once an $SPCQ_i$ is submitted, it will stay at C_i and be executed periodically from its begin time ($begin_i$) to end time (end_i) which is defined to be its begin time plus the time period T_i, e.g. ($end_i = begin_i + T_i$) before C_i arrives its destination, $dest_i$. T_i is a pre-defined system parameter. For simplicity, we assume that $SPCQ_i$ is invoked for execution when C_i is approaching a *new* region before arriving its destination.

The broadcast server is responsible for broadcast of traffic data to the clients through a broadcast channel. In data broadcast, the traffic server selects the traffic data from the traffic database and puts them into a broadcast buffer. The broadcast server, which directly connects to the traffic server, obtains the data from the broadcast buffer and then broadcasts the data one by one according to a defined broadcast schedule. It is assumed that a simple *flat broadcast disk* is adopted in data broadcast, i.e., each broadcast cycle consists of all the data maintained at the traffic server [1]. Once the broadcast server finishes a broadcast cycle, it will redefine the next broadcast schedule.

Each broadcast cycle consists of two parts as shown in Fig. 3. The first part is the broadcast index in which the locations (e.g., the broadcast times) of the data in the broadcast cycle are defined. Following [17], a pre-processed indexing technique is adopted to prepare the broadcast index in which the path information for clients at different regions to their destination regions are included in the index. With the index, each client may only need to capture a sub-set of traffic data for performing shortest path search, e.g., required regions in *NR*.

The second part of the broadcast cycle, called network data part, is the values of traffic data indicating the connections of the road segments and their weights. In order to reduce the energy cost for getting traffic data from the broadcast channel, a client first reads the broadcast

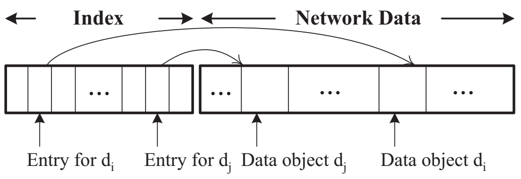


Fig. 3. Broadcast cycle.

index. After identifying the broadcast times of its required data as specified in the broadcast index, it will switch to doze mode of operation to conserve energy and wake up just before the broadcast of its required data.

After obtaining all its required traffic data, the client constructs a local road network and then invokes its *SPCQ* to search the shortest path from its current location using a shortest searching algorithm, e.g., the Dijkstra algorithm. Then, the client will follow the obtained shortest path which consists of a sequence of directed edges to be navigated to its destination. It is assumed that the clients have limited energy and computation power. Therefore, minimizing the tune-in cost for getting road data from data broadcast and the path searching cost for each invocation of *SPCQ* are the main performance concerns in this study.

While traffic data are being broadcast to the clients, traffic updates may arrive at the traffic server for refreshing the current values of the traffic data in the traffic database. If an *SPCQ* accesses to outdated traffic data for path searching, the obtained shortest path could be incorrect. In order to maintain the consistency of the set of traffic data, it is assumed that the installations of updates arrived during a broadcast cycle are deferred until the end of the broadcast cycle [19,31]. How to improve the data freshness to the clients with the uses of different update models and broadcast scheduling algorithms are important future works.

4. The problems and motivation examples

In this section, we discuss the navigation problems in processing an *SPCQ* on a road network. In the example, we use *NR*, which has been shown to be one of the best pre-processed indexing method to reduce the costs at the clients, to illustrate the traffic update problem on the shortest path search and re-generation of broadcast index. Note that *NR* generates sets of required regions for clients at different regions to perform shortest path search to obtain the paths to their destinations. As shown in Fig. 4, the road network is divided into non-overlapping regions and each region has a set of *border vertices* represented by black circles as shown in Fig. 4(a). A border vertex is a vertex that has at least one neighbor lying in a different region. A border vertex entering into a region is called an *in-vertex* while a border vertex moving out a region is called an *out-vertex*. For each region pair (called begin region and end region), the traffic server generates a set of *required regions*, e.g., the gray blocks in Fig. 4(a). Then, a client gets the traffic data of its required regions according

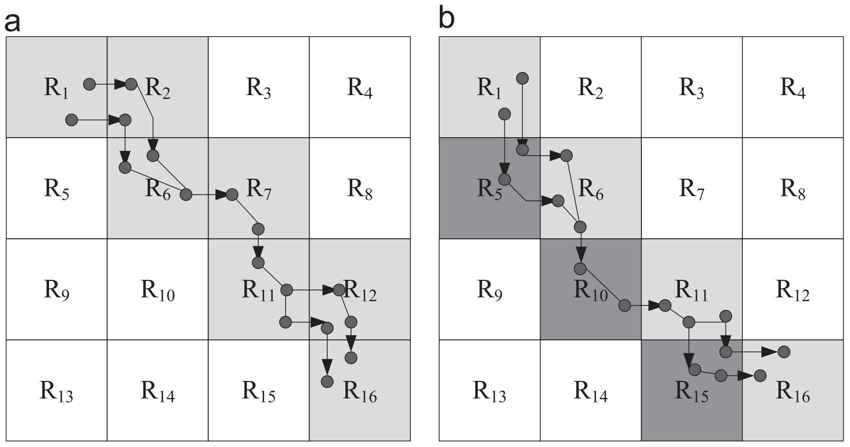


Fig. 4. Examples of required regions in NR. (a) Before the updates. (b) After the updates.

to the information maintained in the broadcast index to calculate the shortest path from its current location to its destination.

4.1. *Problem 1: frequent index re-generation at traffic server*

Updating the traffic data could affect the set of *required regions* for each pair of regions. For example, due to traffic updates, the shortest path for going from region R_1 to region R_{16} shown in Fig. 4(a) changes to the one shown in Fig. 4(b). Therefore, the sets of required regions have to be re-calculated. Although the values for the index associated with R_1, R_6, R_{11} and R_{16} remain the same as they are required regions before and after the updates, for indices of the remaining regions, the required regions are different. After a new set of required regions is identified for a region pair, the broadcast index has to be re-generated accordingly. Note that the shortest path information for all the region pairs has to be recomputed after each batch and the resulted re-computation cost can be expensive if the number of regions is large.

4.2. *Problem 2: changes in the shortest path*

Another serious performance problem of *NR* and other similar shortest path searching techniques when they are applied for *SPCQ* in navigation on a road network is the heavy re-calculation cost at the clients and many of them may not be necessary since the road traffics are changing. Note that even with *NR*, the searching cost for the shortest path over the sub-graph generated from the required regions could still be expensive as the clients have limited processing capabilities, especially when the destinations are far away from their current locations.

For example, as shown in Fig. 5, the client is currently at location *s* and moving towards destination *t*. Based on the current road traffic, the shortest path obtained at location *s* is a set of edges in the road network, $e_1, e_2, \ldots, e_n$ (drawn in thick lines). When the client arrives location *a*, a new shortest path is obtained to direct the client to move through location *c* to location *f* and then location *g* (in dash

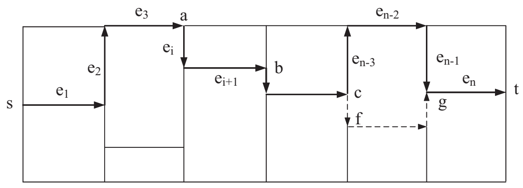


Fig. 5. An illustration of the navigation problems in *SPCQ* with traffic updates.

lines), instead through the previous route e_{n-3}, e_{n-2}, and e_{n-1}. Since the client is still at location *a*, the change in the shortest path actually does not affect its current movement and further changes in the path may occur later due to changes in road traffic. On the other hand, the cost for the computation could be expensive if the distance between the current location and the destination is large. Note that although various prediction techniques have been proposed to estimate the future traffic of roads, it is still an open problem to obtain high accuracy in road traffic prediction for the road networks of big cities which have heavy traffic. Although extending the period for re-execution of *SPCQ* may be able to reduce the computation cost of the queries at the clients, this could seriously increase the probability of delaying the arrival times of the clients to destinations. Even worse, a client may be "blocked" within a road segment if it keeps following the current outdated path due to some "unexpected traffic" jam, e.g., off-loading of goods from a vehicle and traffic accident.

5. Progressive approach (PA)

In this paper, we propose the *progressive approach* (*PA*) for execution of *SPCQ* at mobile clients. The main performance goal of *PA* is to minimize the tune-in and path searching costs at the clients without seriously delaying their arrival times to destinations. In *PA*, to reduce the path searching cost at the clients, similar to *NR*, a pre-processing is performed at the traffic server to define a set of required regions for a client to build its local graph. In addition, a hierarchical indexing technique is adopted by the clients to monitor the shortest paths in their local graphs. As shown in the previous example, many road segments in the latter parts of the current shortest path may not be used by the clients due to changing paths. Therefore, in *PA*, higher levels of connections, called *shortcuts*, are defined for connecting locations which are farther away from the current location of a client to reduce the searching cost. A shortcut consists of one or several road segments showing the shortest travel time between an out-vertex and an in-vertex of the regions in the road network. In Section 6, we will give a detail discussion on shortcuts.

As shown in Fig. 6, the main idea of the approximation technique adopted in *PA* is that it *progressively* generates the shortest path information from each invocation of an *SPCQ* to direct the client to the destination *step-by-step* while it is following the suggested path to the destination. In each invocation, the returned shortest path consists of the road segments of the current region of the client as well as different levels of shortcuts showing the shortest distances to the destination region. The road segment of the current region is used to determine the current movement of the client while a higher level shortcut is provided for going to a vertex which is farther away from its current region. A more detailed path to the destination will be generated when the client is approaching the destination.

Fig. 7 summarizes the six main steps of *PA*. The details of the steps will be discussed in Sections 6 and 7.

1. The regions in the road network are first organized into a hierarchy, called a *region tree*, and then the border vertices in each region in the hierarchy are connected directly by shortcuts. The network formed by the shortcuts in a region is called a *shortcut network* (*SN*) of the region.
2. From the hierarchy of *SN* formed from the regions in the region tree, we formulate a *multi-level shortcut network (MLSN)* for each pair of regions (begin region and end region) in the original road network. In an *MLSN*, different levels of *SN* are selected according to their locations in the region tree.
3. A shortest path search is performed on each *MLSN* to prune the set of *SN* in which the shortest path will not pass through. The *pruned MLSN* is used to generate a client *MLSN*, $MLSN_C$, after considering the distance from the current region of the client.
4. The broadcast index is generated according to $MLSN_C$ to include the shortest path information for clients going at different regions to their destinations. The broadcast index and the network data including *SN* of $MLSN_C$ form the broadcast cycle.
5. The clients obtain the broadcast index, which contains the broadcast times of *SN* of their $MLSN_C$, and then they capture the corresponding *SN* according to the information maintained in the broadcast index.
6. According to the *SN* in its $MLSN_C$, a client road network (RN_C) is constructed and a shortest path search is performed on the client road network to obtain the shortest path information for the client to go to its destination.

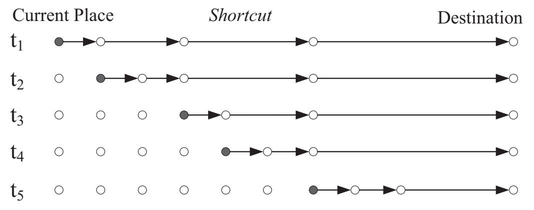


Fig. 6. An illustration of the main idea of the progressive approach (PA).

In addition to minimizing the searching cost at clients, another important property of *PA* is that it is easily extended to reduce the index re-generation cost at the traffic server by defining an *update threshold* for each level of *SN* to prune out "unimportant traffic updates". These are the updates that will not "significantly" affect the current shortest path information to be distributed to the clients. Note that an *SN* is a consolidated network formed by merging road segments. The shortcuts in an *SN* may contain higher errors on the path length especially for the shortcuts at higher level *SN*. The pruning mechanism will be discussed in Section 6.

6. Shortcut networks (SN)

6.1. Region tree

The first step of *PA* is to merge neighboring regions in the road network into higher level regions to form a *region tree* as shown in Fig. 8. In the region tree, each leaf node is a region *R* (or R^1 for level 1) and each inner node at level *i* with $i>1$ is a higher level region R^i. We use r^i to denote

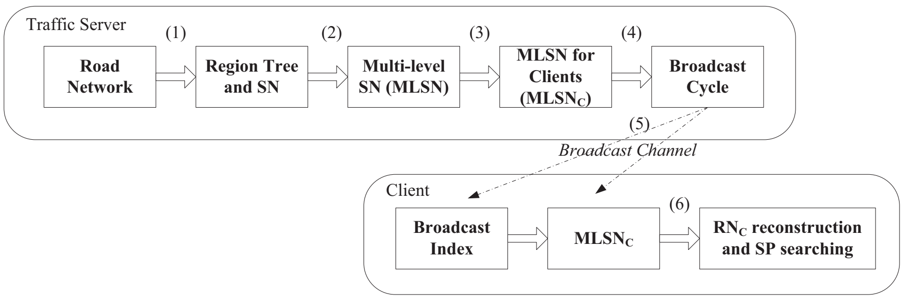


Fig. 7. The main steps in *PA*.

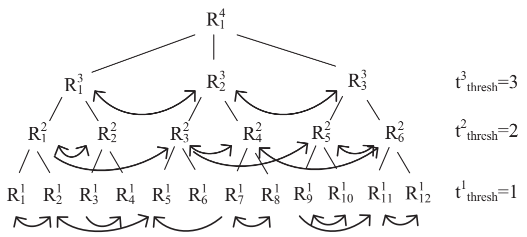
the number of regions/higher level regions at level i. Let $children(R^i)$ be the set of R^{i-1} merged under R^i, and reversely, $parent(R^i)$ be R^{i+1} such that $R^i \in children(R^{i+1})$. $Sibling(R^i)$ is defined to be $children(parent(R^i)) \backslash R^i$.

It is assumed that the number of child nodes under an inner node (called the branching factor), i.e., the number of R^i to be grouped under R^{i+1}, in a region tree is a pre-defined system parameter. Since the focus of this paper is on how to efficiently process *SPCQ* instead to find an optimal way to organize the regions into higher level regions to maximize the system performance, we will leave this as an important future work. For simplicity, we assume that the branching factor at each level is the same.

Fig. 8 shows an example of a region tree with 4 levels. Note that the regions at the same level may be connected through border vertices as shown in Fig. 8 according to the connections of the road segments in the original road network. In addition, as shown in Fig. 8, for each level, an update threshold is defined to determine under which condition an update should be installed or be pruned in maintaining the *SN* of the regions at the level. The values of the update thresholds at higher levels are larger since the uncertainties of the weights of the shortcuts for a higher level region are usually greater, i.e., due to a larger road network to be represented by the *SN*.

6.2. Generations of SN

In order to simplify the road network, i.e., to reduce the number of vertices and edges, shortcuts are created for each region at different levels of the region tree. A *shortcut* is a *directed edge* with a *weight* to indicate the travel time between two border vertices of a region/higher level region instead of showing *how* to go from one road segment to another road segment within a region.



R_1^4
R_1^3 R_2^3 R_3^3
R_1^2 R_2^2 R_3^2 R_4^2 R_5^2 R_6^2
R_1^1 R_2^1 R_3^1 R_4^1 R_5^1 R_6^1 R_7^1 R_8^1 R_9^1 R_{10}^1 R_{11}^1 R_{12}^1
$t^3_{thresh}=3$
$t^2_{thresh}=2$
$t^1_{thresh}=1$

Fig. 8. An illustration of region tree.

We first convert each lowest level region (a level 1 region) in the region tree to be an *SN* (also called level 1 *SN*) by directly creating a shortcut between an in-vertex and an out-vertex of the region to indicate the *shortest path* between the two vertices. The weight $(W(S_i))$ of a shortcut (S_i) is the sum of the total travel time (w_i) of each road segment (r_i) on the shortest path connecting the two vertices. In a special case, a shortcut can be a road segment if it is exactly the shortest path.

For example, Fig. 9(a) is the original road networks of $R_1, \ldots, R_4$, i.e., $Or(R_i^1)$ of $i \in \{1,2,3,4\}$. The in/out-vertices are represented by black circles while the starting and ending points of road segments are represented by white circles. Between the vertices, solid lines are the road segments in the original road network and dash lines are shortcuts. Fig. 9(b) shows the examples *SN* for the lowest regions R_i^1, $SN(R_i)$ of $i \in \{1,2,3,4\}$, figured in each square. Note that in an *SN*, a shortcut is created between two vertices *only if* the distance between them is the shortest. Otherwise, the number of shortcuts in the region could be large and the *SN* for the region could be complex. For example, $SN(R_1)$ in Fig. 9(b) contains three shortcuts. The shortcut (a,c) is required because it can shorten the distance between a and c from 7 to 6. If the shortest path is 7, (a,c) could be removed.

After creating an *SN* for each lowest level region, we create *SN* for higher level regions. Let $\bigcup_{1 \le i \le r} SN(R_i)$ be the union of the *SN* for each region in the set $\{R_1, \ldots, R_r\}$ which are the regions at the same levels. The construction of *SN* for a higher level region can be conducted on the union of *SN* for its children in the region tree, instead of on the original road network, as the union of *SN* is usually simplified. For example, Fig. 9(b) is $\bigcup_{1 \le i \le 4} SN(R_i)$, the *SN* formed by the union of $SN(R_i)$ with $i \in \{1,2,3,4\}$, and Fig. 9(c) is the *SN* for the two level 2 regions, figured in each square.

Fig. 10 shows examples of *SN* for the regions at different levels of the region tree shown in Fig. 8. Note that in the figure, in each region, only the border vertices are shown to make the figure easier to read.

6.3. Maintenance of SN

Algorithm 1. Maintaining SN (updates of road segments).

```
1:  Map all updates into their corresponding regions $R$;
2:  Let $C = \{R_j | R_j$ have updates$\}$ and $P = \varnothing$;
3:  $i = 1$;
4:  while $C \ne \varnothing$ and $i$ is not the root level do
5:    for each $R_j^i$ in $C$ do
```

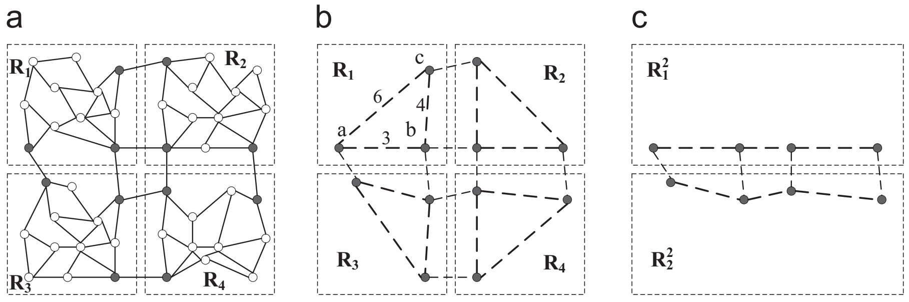


Fig. 9. Examples of the road network and *SN*. (a) Original network. (b) $\bigcup_{1 \le i \le 4} SN(R_i)$. (c) $SN(R_1^2) \cup SN(R_2^2)$.

```
6:       if $i==1$ then
7:         Calculate the new shortest paths $SP$ between border
         vertices in $Or(R_j)$;
8:       else
9:         Calculate the new shortest paths $SP$ between border
         vertices in $\bigcup_{R_k^{i-1} \in children(R_j^i)} SN(R_k^{i-1})$;
10:      end if

11:      Set update flag $F=false$;
12:      for each possible in/out vertex pair do
13:        if there is a shortcut $SC$ in $SN(R_j^i)$ then
14:          if $SP$ contains no border vertex then
15:            if $|W(SP)-W(SC)| > t^i_{thres}$ then
16:              Delete $SC$ and add a new shortcut for $SP$.
17:              $F=true$.
18:            end if
19:          else
20:            Delete the shortcut $SC$.
21:          end if
22:        else if $SP$ contains no border vertex
23:          Add a new shortcut for $SP$.
24:          $F=true$.
25:        end if
26:      end for
27:      if $F==true$ then
28:        Add $parent(R_j^i)$ into $P$.
29:      end if
30:    end for
31:    $C=P$ and $P=\varnothing$;
32:    $i++$;
33: end while
```

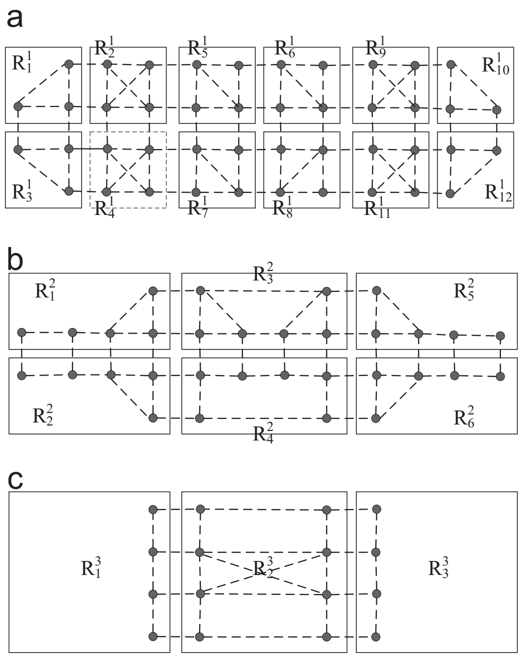


Fig. 10. The shortcut networks for the region tree shown in Fig. 8. (a) $\bigcup_{1 \le j \le r^1} SN(R_j^1)$. (b) $\bigcup_{1 \le j \le r^2} SN(R_j^2)$. (c) $\bigcup_{1 \le j \le r^3} SN(R_j^3)$.

In addition to simplifying the road network, another important goal of *SN* is to reduce the number of traffic updates to be installed for each region including the regions at higher levels. Updating a region may affect the shortest path between two regions, and require re-generation of the broadcast index.

We adopt a *bottom-up* approach for installing the traffic updates into the regions, as defined in Algorithm 1. Note that the updates are installed after the finish of a broadcast cycle and they are the updates arrived during the broadcast cycle. To install the updates, firstly, all the updates are mapped to their corresponding regions at the lowest level, R_j^1 (Line 1). Then, the traffic server performs the shortest path search on the original road network of a region R_j^1 after including the new updates for the region to obtain a new shortest path between the in/out-vertex of R_j^1 (Line 7). Then, for each in/out-vertex pair of R_j^1, if there is a shortcut connecting them and the total weight (i.e., travel time) of the new path connecting the two vertices is not smaller or larger than the weight of the current shortcut connecting them by the update threshold t^1_{thres} for the level, i.e., level 1, we will keep the current shortcut and the updates for the region will be *pruned* as they do not significantly change the weight of the shortcut connecting the two vertices. Otherwise, we replace the current shortcut by the new shortcut after installing the update for connecting the two vertices (Line 16). If there is a shortcut between the two vertices but there is another border vertex on the new path, we can simply delete the current shortcut (Line 20), because it will not shorten the distance between the two vertices. If currently there is no shortcut connecting the two vertices, we will first check whether a shortcut is needed after installing the updates (Line 22) and add the shortcut if needed (Line 23). That is to check whether the new shortcut shortens the distance between the two vertices in $SN(R_j^1)$. In implementation, an equivalent way is to check if the new shortest path to be represented by the shortcut contains another border vertex. If it contains another border vertex, the shortcut will not shorten the distance between the vertices in the *SN*. This finishes the updates of the set of *SN* at the lowest level, i.e., $\bigcup_{1 \le j \le r^1} SN(R_j^1)$.

If after installing all the updates to *SN* for the lowest level regions, $\bigcup_{1 \le j \le r^1} SN(R_j^1)$ does not change, i.e., the set of shortcuts remains the same, the update process will be stopped and the index will not need to be regenerated as there are no changes in the shortest paths for the regions. Otherwise, we go to maintain the *SN* for higher level regions, i.e., $\bigcup_{1 \le j \le r^i} SN(R_j^i)$ for $i>1$ and update their *SN*. The updates of higher level *SN* are similar to the lowest level except that the shortest path search is performed in $\bigcup_{R_k^{i-1} \in children(R_j^i)} SN(R_k^{i-1})$ (Line 9), instead of $Or(R_j^i)$, since the former one is simpler.

An example of maintaining *SN* is illustrated in Fig. 11. Fig. 11(a) is the original road network $Or(R_5^1)$ of R_5^1 and Fig. 11(b) is the corresponding shortcut network $SN(R_5^1)$ shown in Fig. 10. The shortcut (c,j) in $SN(R_5^1)$ in Fig. 11(b) represents the shortest path $\langle c,g,j \rangle$. Suppose there is an update to change the travel time of edge (g,j) from 2 to 4. The new shortest path between them is still $\langle c,g,j \rangle$ after the update but with total time of 7. Since the difference in travel time of the new shortest path and the original path is larger than $t^1_{thres}=1$, we go on to process it in $\bigcup_{R_k^1 \in children(R_3^2)} SN(R_k^1)$ as in Fig. 11(c). Because the update of (c,j) from 5 to 7 does not change any shortest path between in/out-vertices of R_3^2, $SN(R_3^2)$ remains the same

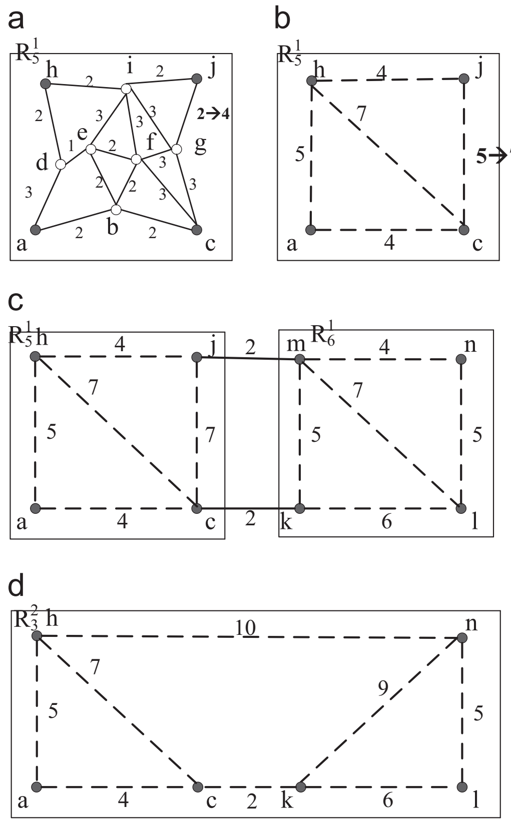


Fig. 11. Maintaining shortcut network example. (a) $Or(R_5^1)$. (b) $SN(R_5^1)$. (c) $\bigcup_{R_k^1 \in children(R_3^2)} SN(R_k^1)$. (d) $SN(R_3^2)$.

as shown in Fig. 11(d). Then, the update of *SN* is completed and will not be propagated to higher level regions and *SN*.

7. Multi-level shortcut networks (MLSN)

7.1. Generation of MLSN

7.1.1. MLSN for traffic server

After creating different levels of *SN* from the regions in the region tree, we generate the multi-level shortcut networks (*MLSN*) by merging the different levels of *SN* at the traffic server for clients moving from different regions to their destination regions (called begin region and end region, respectively). The *MLSN* for a pair of regions is generated by considering the positions of the begin region and end region in the region tree.

Consider all the lowest level *SN*, *SN* of the begin region and end region are selected first for the path searching. Other lowest level *SN* can be *replaced* by higher level *SN*, if the regions of the lowest level are children of the higher level region. The replacement is recursively applied until a higher level region that due to further replacement would contain the begin region or end region as a descendant, leading to an overlapping. Overlapping of *SN* could enlarge the size of a *MLSN*. Note that a higher level *SN* covers a larger area and is simpler compared with the original road network. The resulting *MLSN* is a simplified network of the entire road network.

Denoted by $MLSN(R_s^1, R_t^1)$ the *MLSN* for the pair of regions (R_s^1, R_t^1). Given a pair of regions (R_s^1, R_t^1), Algorithm 2 shows how to generate $MLSN(R_s^1, R_t^1)$ from the region tree. In Algorithm 2, it first finds the lowest common ancestor *Lca* of R_s^1 and R_t^1 in the region tree (Line 2). Then, for each R^i on the path from R_s^1 (R_t^1, respectively) to *Lca*, it merges the *SN* of the sibling of R^i into $MLSN(R_s^1, R_t^1)$ if it is not on the path from R_t^1 (R_s^1) to *Lca* (Lines 4–9). Finally, for each R^i on the path from *Lca* to the root, it adds the *SN* of the sibling of R^i into $MLSN(R_s^1, R_t^1)$ (Lines 11–14).

Algorithm 2. Generating MLSN for traffic server.

```
Input: a region pair $(R_s, R_t)$
Output: $MLSN(R_s, R_t)$
1:  Let $C_s = R_s$, $C_t = R_t$;
2:  Let $Lca = LCA(R_s, R_t)$;
3:  $MLSN(R_s, R_t) = \{R_s, R_t\}$;
4:  while $C_s \neq Lca$. do
5:      For each $R^k \in sibling(C_s) \backslash C_t$, merge $SN(R^k)$ into $MLSN(R_s, R_t)$.
6:      For each $R^k \in sibling(C_t) \backslash C_s$, merge $SN(R^k)$ into $MLSN(R_s, R_t)$.
7:      $C_s = parent(C_s)$.
8:      $C_t = parent(C_t)$.
9:  end while
10: Let $C = Lca$;
11: while $C \neq Root$ do
12:     For each $R^k \in sibling(C)$, merge $SN(R^k)$ into $MLSN(R_s, R_t)$.
13:     $C = parent(C)$.
14: end while
15: return $MLSN(R_s, R_t)$;
```

For example, in Fig. 12(a), $MLSN(R_2^1, R_{11}^1) = SN(R_1^1) \cup SN(R_2^1) \cup SN(R_2^2) \cup SN(R_2^3) \cup SN(R_5^2) \cup SN(R_{11}^1) \cup SN(R_{12}^1)$, where *SN* of higher level region R_2^2 are used to replace the lowest level regions R_3^1 and R_4^1 while R_5^1, R_6^1, R_7^1 and R_8^1 are simplified to *SN* of region R_3^2 at a higher level. For region pair of (R_3^1, R_4^1), $MLSN(R_3^1, R_4^1) = SN(R_1^2) \cup SN(R_3^1) \cup SN(R_4^1) \cup SN(R_2^3) \cup SN(R_3^3)$ as shown in Fig. 12(b). Note that $SN(R_2^3)$and $SN(R_3^3)$ are still required to ensure the correctness of the shortest path.

7.1.2. MLSN for clients ($MLSN_C$)

After creating the *MLSN* for each pair of regions in the original network, the traffic server searches the shortest path in each *MLSN*. Then, the traffic server prunes out those *SN* not on the shortest path from each *MLSN*. The remaining *SN* forms the *pruned MLSN*.

Given a pruned MLSN for a region pair, we select *SN* from different levels to generate the *client* multi-level shortcut network, $MLSN_C$, for the region pair after considering the distance from the current region of the client as defined in terms of the number of lowest level regions. The original road network $Or(R)$ is selected for the current region R of the client as the client needs to have the detail road information in navigation to the out-vertex of the region. *SN* for higher level regions are selected for regions closer to the end region.

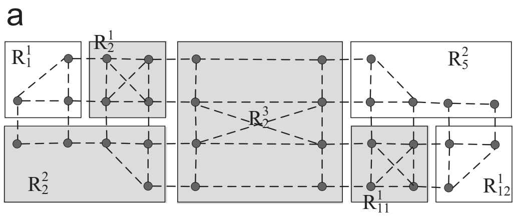
Denoted by $MLSN_C(R_s, R_t)$ the client multi-level shortcut network $MLSN_C$ for region pair (R_s, R_t). As in example in Fig. 12(a), the traffic server obtains the shortest path which does not pass through R_1^1, R_5^2 and R_{12}^1. Therefore, the corresponding *SN* will be pruned. The pruned *MLSN* is $SN(R_2^1) \cup SN(R_2^2) \cup SN(R_2^3) \cup SN(R_{11}^1)$, colored in gray. After considering the distance from the current region R_2^1,

$MLSN_C(R_1, R_{11})$ is formulated as $Or(R_2^1) \cup SN(R_2^2) \cup SN(R_2^3) \cup SN(R_3^3)$ as shown in Fig. 13(a). In Fig. 12(b), the traffic server finds that the shortest path only passes through $\{R_3^1, R_4^1\}$ whose *SN* are colored, and then the traffic server generates $MLSN_C(R_3, R_4) = Or(R_3^1) \cup SN(R_4^1)$, as shown in Fig. 13(b).

Note that $MLSN_C$ may not contain the road segment of the destination. It may only contain the border vertices of the region/higher level region containing the road segment of the destination. Therefore, the client, after capturing the corresponding $MLSN_C$ (encoded in the broadcast index in Section 7.2) and reconstructing its own local road network, the client road network RN_C, will perform the shortest path search to find a path leading to the *nearest border vertex* of destination region. The detailed path to the destination road segment will be determined when the client arrives its destination region.

7.2. Generation of broadcast index

After constructing $MLSN_C$ for each pair of regions, the traffic server formulates the broadcast index. A broadcast cycle starts with the generated broadcast index, followed by network data, as illustrated in Fig. 14. The index consists



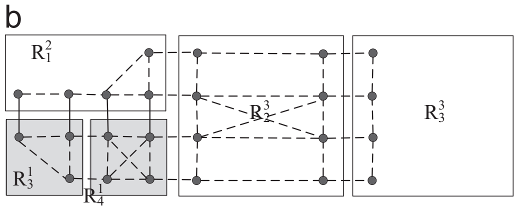
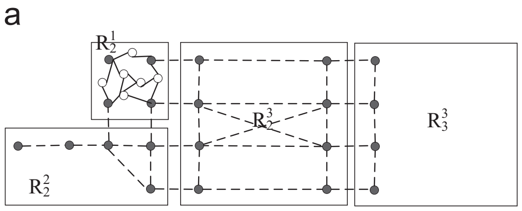


Fig. 12. An example of searching the shortcut network. (a) $MLSN(R_2^1, R_{11}^1)$. (b) $MLSN(R_3^1, R_4^1)$.



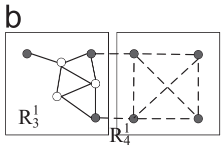


Fig. 13. $MLSN_C$ for the example in Fig. 12. (a) $MLSN_C(R_2^1, R_{11}^1)$. (b) $MLSN_C(R_3^1, R_4^1)$.

of a kd-tree partitioning information and a table. The kd-tree partitioning information includes the values adopted in the partitioning and can be used to map a location to the region containing it. The table specifies, for each pair of regions (R_s, R_t), the components (*SN* or *Or*) of $MLSN_C(R_s, R_t)$ and the broadcast schedule (times) for each of them.

We adopt the common arrangement of placing the broadcast index at the beginning of a broadcast cycle, different from the (1,*m*)-interleaving or variant proposed in *NR* (i.e., the index for each region is different and placed in front of the region's data). Unlike the shortest path query to be executed for once, the access latency for broadcast data is not a critical performance measure in processing *SPCQ* as the client has already obtained a path to follow. The main delay for movement is the latency for the first set of path information.

The network data consists of the *SN* of different level regions and the original network *Or* of the lowest level regions. Different organizations of them in a broadcast cycle may have different performances in practice under different workloads. It is not the focus of this paper to study how to schedule the broadcast data to achieve the maximum performance, e.g., minimizing the waiting time for getting all traffic data by a client. For simplicity, we assume the use of the flat broadcast disk to formulate the broadcast schedule and broadcast the *SN* and *Or* in a pre-ordered traversal order of the region tree. For each traversed higher level region, its *SN* is placed in the cycle while for each traversed lowest level region, both *SN* and *Or* are included in the broadcast cycle. For example, as in Fig. 14, the broadcast sequence of the network data is $SN(R_1^3)$, $SN(R_1^2)$, $SN(R_1^1)$, $Or(R_1^1)$, $SN(R_2^1)$, $Or(R_2^1)$, etc. for the region tree shown in Fig. 8.

7.3. Processing of SPCQ at clients

The execution of an *SPCQ* at a mobile client is summarized in Algorithm 3. Each query $Q_i(cp_i, dest_i, begin_i, end_i)$ issued by client C_i will be executed during the time period from $begin_i$ to end_i if C_i has not arrived the destination $dest_i$ as in Line 1. In each invocation, C_i first reads the next index in the broadcast cycle from its first tune-in to the broadcast channel, and obtains $MLSN_C$ according to its current location cp_i and destination $dest_i$. Then all the *SN* in $MLSN_C$ will be captured by C_i according to the broadcast schedule defined in the broadcast index. After that, a client road network RN_C is constructed by C_i and a shortest path search is performed on RN_C with cp_i and $dest_i$ to obtain a path *P*. Then, C_i follows *P* until it is approaching a new region or $dest_i$. Then another invocation of the *SPCQ* will be performed or the query is terminated if it has arrived $dest_i$ or the current time equals to end_i.

Algorithm 3. Query algorithm at client.

```
Input: Query $Q_i(cp_i, dest_i, submit_i, end_i)$
Output: Client $C_i$ arrives $dest_i$ before $end_i$ or current time=$end_i$
1: while $C_i$ does not arrive $dest_i$ or the time is before $end_i$ do
2:     Client $C_i$ receives the next index from the broadcast cycle;
3:     Based on the current location $cp_i$ and $dest_i$, $C_i$ retrieves the
       $MLSN_C$.
4:     According to the broadcast time of each SN in $MLSN_C$, $C_i$ sleeps
       until the SN is broadcast. This process continues until all SN in
       $MLSN_C$ are captured.
5:     $C_i$ constructs client road network $RN_C$ and performs shortest
       path search on $RN_C$ to obtain a path P.
```

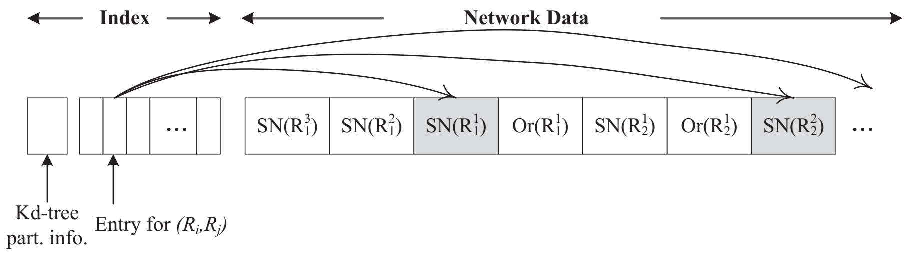


Fig. 14. An illustration of broadcast cycle organization in PA.

```
6:    $C_i$ follows $P$ until it is approaching a new region or has arrived
      $dest_i$ or current time=$end_i$.
7: end while
```

7.4. Cost analysis

In this section, we give the cost analysis of *PA* as compared with *NR* and *HiTi*. In the analysis, we concentrate on the following two performance measures:

1. The index re-generation cost at the traffic server. In *NR* and *PA*, the shortest path computation is the dominating cost in generating the broadcast index. In the analysis, to simplify the discussion, we assume to use the Dijkstra algorithm for searching the shortest path. More advanced algorithms may accelerate them in path searching. The cost of the searching algorithm is quantified by the *search space*, i.e., the number of settled vertices and the sum of degrees of settled vertices (i.e., the number of relaxed edges).
2. The broadcast cycle size, including the index size and the size of network data. A larger broadcast cycle implies longer waiting time to get the required data and more network bandwidth to be consumed for data broadcast.

Let $Adj(G)$ be the size of the original road network G in an adjacency list representation, which is a suitable linear function of the number of vertices and edges in G. Let $Dij(G)$ be the average number of edges traversed (or called "relaxed edges" in the rest of the paper) during a Dijkstra search in G. Let $Bor(R^i)$ be the number of border vertices in R^i.

The index re-generation cost of *PA* is

$$\sum_{1\le j\le r} Bor(Or(R_j)) * Dij(Or(R_j)) + \sum_{2\le i\le l}\sum_{1\le j\le r^i} Bor\left(\bigcup_{R_k^{i-1}\in children(R_j^i)} SN(R_k^{i-1})\right) * Dij\left(\bigcup_{R_k^{i-1}\in children(R_j^i)} SN(R_k^{i-1})\right) + \sum_{1\le i\le r}\sum_{(1\le j\le r\wedge j\ne i)} Bor(Or(R_i)) * Dij(MLSN(R_i,R_j)),$$

where the first two terms are the costs for maintaining the *SN* in the lowest level, and higher levels, respectively. The last term is the cost of searching the *MLSN* for pruning out the *SN* that are not on the shortest path.

In comparison, the index re-generation cost of *HiTi* is the maintenance cost of the *SN* in the hierarchy and *HiTi* does not need to perform the computation for pruning unnecessary *SN*. Formally, it is

$$\sum_{1\le j\le r} Bor(Or(R_j)) * Dij(Or(R_j)) + \sum_{2\le i\le l}\sum_{1\le j\le r^i} Bor\left(\bigcup_{R_k^{i-1}\in children(R_j^i)} SN(R_k^{i-1})\right) * Dij\left(\bigcup_{R_k^{i-1}\in children(R_j^i)} SN(R_k^{i-1})\right).$$

For *NR*, the index re-generation cost is

$$\sum_{1\le i\le r}\sum_{(1\le j\le r\wedge j\ne i)} Bor(Or(R_i)) * Dij(G).$$

Compared with the cost of *NR*, $Dij(MLSN(R_i,R_j))$ in *PA* should be smaller than $Dij(G)$ in *NR* since $MLSN(R_i,R_j)$ has a smaller size than G. However, *PA* requires an additional cost for maintaining the *SN*. It leaves to Section 8 to see which one has lower index re-generation cost as a whole.

The broadcast cycle size of *PA* is

$$r + r*r*Req_{PA}(G) + Adj(G) + \sum_{1\le i\le l}\sum_{1\le j\le r^i} Adj(SN(R_j^i)),$$

where the first two terms are the index size and the last two terms are the size of network data. The first term is for the kd-tree partitioning information of the road network (linear w.r.t. the number of regions), while the second term is the size of the table indicating $MLSN_C$ for each pair of regions. $Req_{PA}(G)$ is the average number of *SN* of regions or higher level regions in $MLSN_C$ over the total number of region pairs. Because $Req_{PA}(G)\le r$, the index size is in the order of r^3. The last two terms are the size of the original network and the total size of different levels of *SN*, respectively.

For *NR*, the broadcast cycle size is

$$r + r*r*Req_{NR}(G) + Adj(G),$$

where the first two terms are the index size. The second term is the size of the table storing the required regions for each pair of regions. $Req_{NR}(G)$ is the average number of required regions over the total number of region pairs in *NR*. The last term is the size of network data, which is the size of the original network. As $Req_{NR}(G) < r$, the index size is in the order of r^3. However, $Req_{NR}(G)$ is expected to be greater than $Req_{PA}(G)$ as the required regions in *NR* are all lowest level regions.

In general, the network data size of *PA* is greater compared with *NR* since different levels of *SN* are included in a broadcast cycle while the network data of *NR* contains the original road network only. However, the index size of *PA* is usually less than that of *NR*, since $MLSN_C$ in *PA* has smaller cardinalities compared with the total number of required regions in *NR*. (The *SN* of higher level regions may

Table 2
Details of different data sets.

Name	Description	# Vertices	# Edges	Size (in KB)
NY	New York City	264,346	733,846	5291
BAY	San Francisco Bay Area	321,270	800,172	6027

correspond to multiple lowest level regions.) As will be shown in Section 8.1, the difference of the index size between *PA* and *NR* is greater when the number of regions *r* is larger because of more levels. This could make the broadcast cycle size of *PA* to be smaller than that of *NR* as shown in our experimental results in Section 8.1.

The broadcast cycle size of *HiTi* is

$$r + r*r*Req_{HiTi} + Adj(G) + \sum_{1 \le i \le l} \sum_{1 \le j \le r^i} Adj(SN(R_j^i)),$$

which is similar to *PA*. *r* is still required because it adopts the same partitioning method as *PA*. Req_{HiTi} represents the average number of *SN* received for each pair of regions. The value can be obtained from the hierarchy and $Req_{HiTi} < r$.

8. Performance evaluation

The experiments were conducted on a 64-bit machine equipped with Intel Core i7-2600 3.4 GHz CPU and 4 GB main memory. The underlying software platform was Gcc 4.5.1 compiler on OpenSuse 11.4 OS. We used the real road network of North America (downloadable from http://www.dis.uniroma1.it/challenge9/) as the input data set. The details of the data set are referred to Table 2, with BAY as the default network for the experiments. BAY consists of approximately 300 K vertices and 800 K edges. In the implementation, the original road network was essentially in an adjacency list representation and each vertex was assigned 8 Bytes for the vertex identifier and number of incident edges. Each undirected edge was assigned 9 Bytes for the target vertex of an edge, edge weight and edge direction. Two directed edges with the same end vertices and the same weight were merged into an undirected edge, and stored only at the adjacency list of one vertex.

We compared *PA* with *NR*, *HiTi* and *CH*. They were extended for processing of *SPCQ*. The number of lowest level regions *r* in *PA* and *HiTi* was the same as the number of regions in *NR*. The experiments were repeated with different values of *r*, e.g., $r=16$, 64, 128 and 256. For *PA* and *HiTi*, in each level, the number of regions merging into a higher level region *p* was set to be 2 and repeated with $p=4$. For the parameter settings of *CH*, we use the aggressive variant defined in [11,12]. In *PA*, the *SPCQ* was invoked when a client was approaching a new region as the road connections of the new region needed to be acquired. To be consistent, we made *NR*, *HiTi* and *CH* to invoke the *SPCQ* at the same time. After obtaining a new path, the client followed the path step by step to its destination. It was assumed that all the clients could arrive their destinations before the deadlines of their *SPCQ*.

To simulate the changes in traffic, we randomly selected an edge in the original road network and increased or decreased

Table 3
Parameter settings and baseline values.

Parameters	Baseline settings
# Lowest level regions *r*	16, 64, 128, 256
# Regions merged in a higher level region *p*	2, 4
Update range of edge weight *I*	[0,1000]
# Updates *N*	100,000
# Queries *Q*	100
Query distance threshold *d*	2,000,000
Update threshold for the lowest level *t*	0
Update threshold base *m*	2

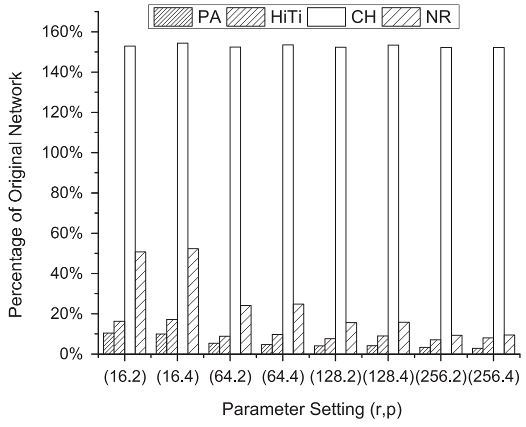


Fig. 15. Size of client road network RN_C.

its weight by a random value within an interval $I=[0, 1000]$. If the weight became a negative value after the update, we simply set it to be zero. After the updates of traffic at the end of a broadcast cycle, the traffic server re-generated the broadcast index.

We defined the update threshold for pruning traffic updates by two parameters *t* and *m*, from which the update threshold for a specific level *i* was calculated as $t_{thres}^i = t*m^{i-1}$. In the first set of experiments, we set $t=0$ and $m=2$ to disable the pruning mechanism. The study on the effects of update thresholds on the performance of *PA* will be discussed in Section 8.4.

8.1. Comparison amongst PA, HiTi, CH and NR

In addition to the performance measures introduced in Section 7.4, we also measured the following four metrics in the experiments:

1. The average actual size of client road network RN_C for each invocation of an *SPCQ*. This affects the tune-in cost for getting traffic data from the broadcast cycle (measured by tuning time), and the memory requirement at the clients for building the local graphs.
2. The path searching cost at a client for executing an *SPCQ*, measured by the average search space of the client road network RN_C.
3. The average travel time ratio of a client to reach the

destination. The travel time ratio of *PA* is defined as the total travel time of a client under *PA* minus the total travel time of the client under *NR* divided by the total travel time of the client under *NR*. Note that the travel times under *NR*, *HiTi* and *CH* are the same since they obtain the exact path to the destination in each invocation of *SPCQ*. An important tradeoff of *PA* for minimizing the searching cost at the clients is longer travel time to destination since the path obtained is a "higher level" path containing shortcuts to the destinations' regions or higher level regions instead of a direct shortest path consisting of road segments to destination.

4. The average number of path changes obtained from re-execution of an *SPCQ*. This is an important indicator in practice, and a small number of path changed invocations that give a different path is preferable.

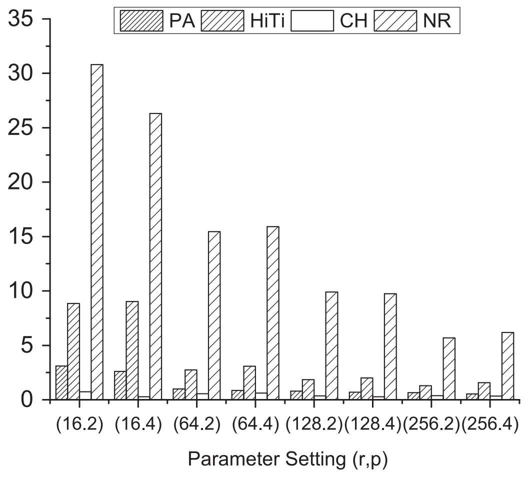


Fig. 16. Searching time at clients (in ms).

The performance results to be reported in below were obtained by averaging over $Q = 100$ randomly generated *SPCQ*, with distances from *cp* to *dest* at the begin time of the query to be larger than a pre-defined threshold $d = 2,000,000$, i.e., the starting location of a client is far away from its destination. In the experiments, we varied the values for *r* and *p*, and fixing $I = [0, 1000]$ and $N = 100,000$ for clarity. The baseline parameter settings are summarized in Table 3.

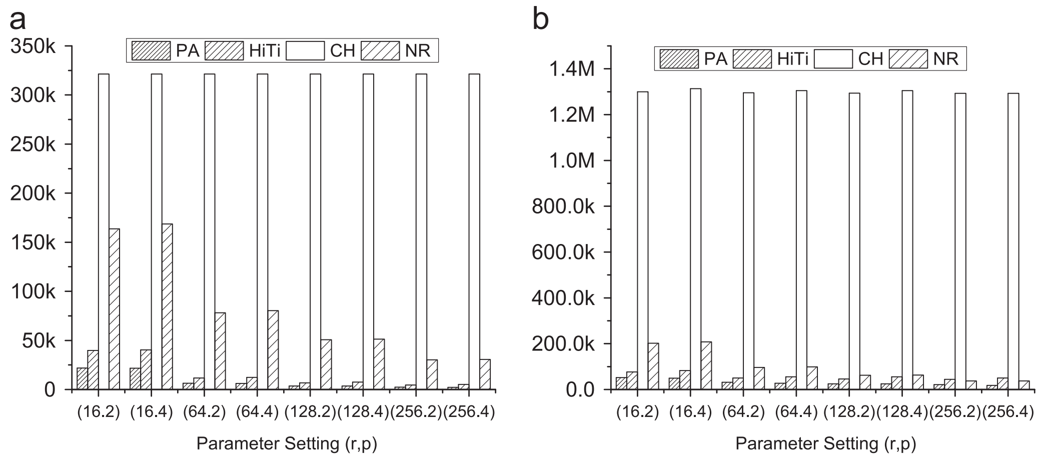


Fig. 17. Details of RN_C. (a) # of vertices. (b) # of edges.

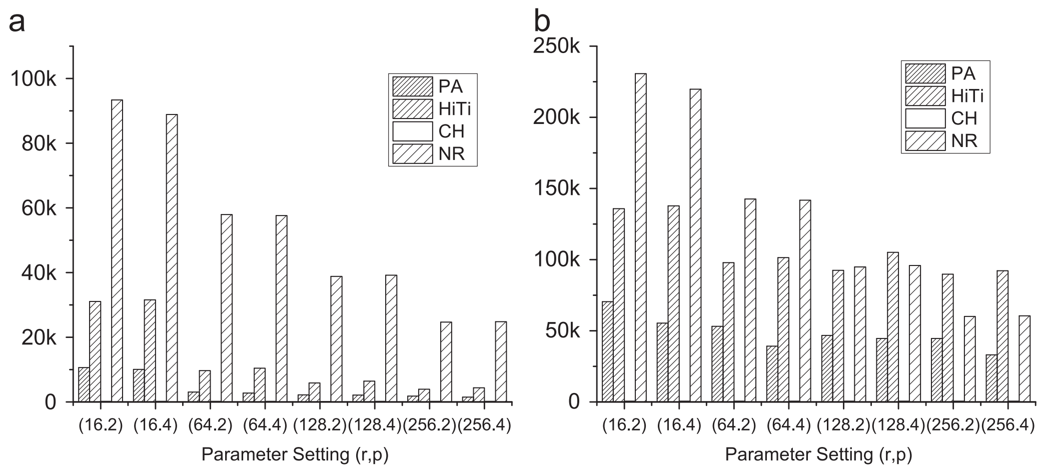


Fig. 18. Search space at clients. (a) # of settled vertices. (b) # of relaxed edges.

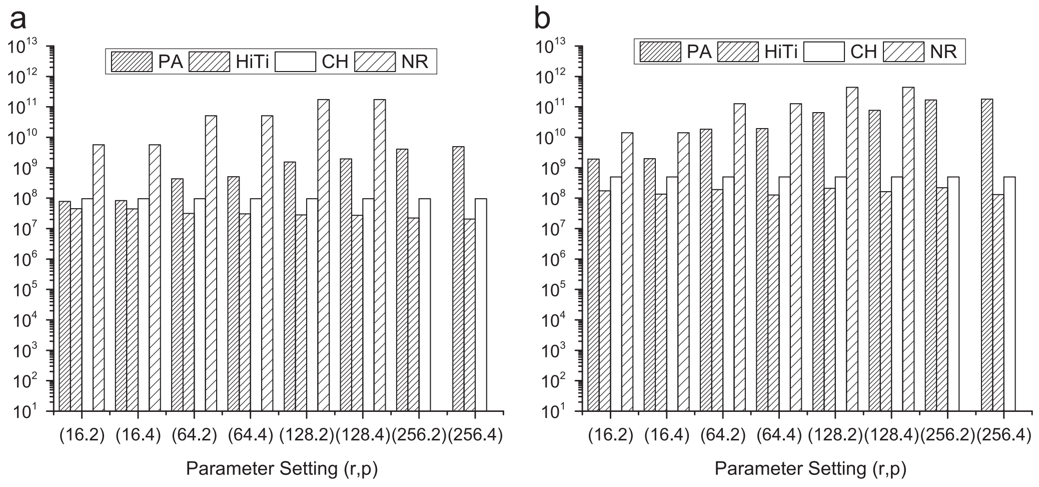


Fig. 19. Search space at traffic server. (a) # of settled vertices. (b) # of relaxed edges.

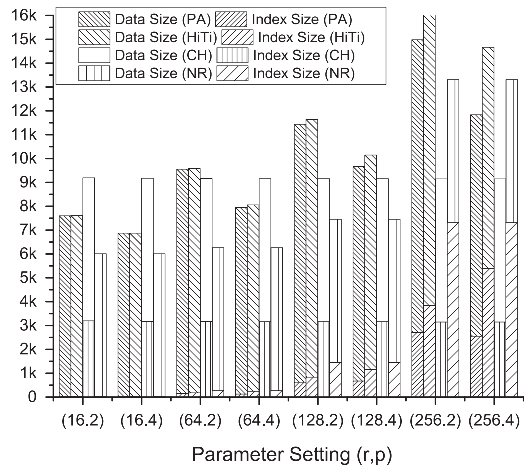


Fig. 20. Size of broadcast cycle and its each part (in KB).

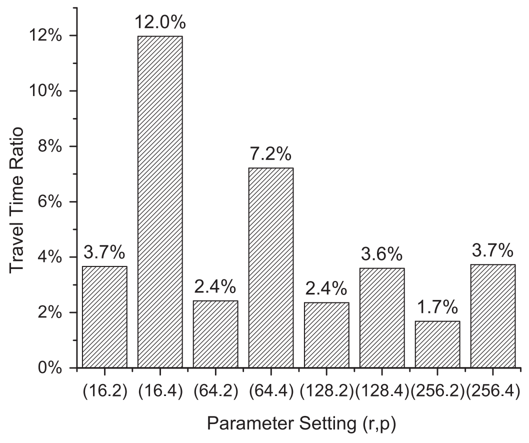


Fig. 21. Travel time to destination (ratio of PA/NR minus 1).

Fig. 15 shows the average size of RN_C of one invocation of an *SPCQ*, normalized to be percentages over the total size of the original road network. As shown in Fig. 15, the three methods (*PA*, *NR* and *HiTi*) can effectively reduce the size of traffic data required by a client for building its client road network to a small portion of the original road network, while *CH* requires the whole structure consisting of the original road network and the shortcuts added to be received by the client. Therefore, *CH* has a very large size of RN_C. For the three methods, consistent with our expectation, the size of RN_C of *PA* is much smaller than that of *NR* and is also consistently smaller than that of *HiTi*. For all the settings of p and r, the size of RN_C of *PA* is just about 25% of *NR* and 50% of *HiTi* on average. When r increases from 16 to 256, the sizes of RN_C of *PA*, *HiTi* and *NR* become smaller due to the finer partitioning while different values of r do not show any significant impacts to the performance of *PA* and *HiTi*. Since the index received by a client is small and negligible compared with the total network data that constitutes RN_C, the tuning time in each invocation of an *SPCQ* can be measured by the size of RN_C. A smaller size of RN_C implies a smaller tuning time at the clients. Therefore, *PA* is more energy efficient for capturing broadcast data compared with *NR* and *HiTi* while *CH* have heavy tune-in cost.

As shown in Fig. 17, although *HiTi* can effectively reduce the number of vertices in RN_C, the reduction in number of edges in RN_C is smaller resulting in a dense network with a small set of vertices. RN_C of *NR* remains a sparse network as it is a portion of the original road network which is mostly sparse. As shown in Fig. 17, by using the pre-processing and approximation techniques, both the number of vertices and edges in RN_C of *PA* are smaller compared with that of *NR* and *HiTi*.

The shortest path searching cost for one invocation of an *SPCQ* depends on the search space. As shown in Fig. 18, *PA* significantly outperforms both *NR* and *HiTi* in reducing the path searching cost. On average, *PA* settles vertices with only 7% of *NR* and 30% of *HiTi*, and relaxed about 40% edges of *NR* and 60% of *HiTi*. This could be a big reduction in energy cost for the periodic executions of *SPCQ* at the clients. We observe that a coarser grouping of regions

($p=4$ compared with $p=2$) lowers the search space at the clients. Consistent with the results shown in Fig. 18, the searching time of *PA* for finding the shortest path at the clients is also smaller than that of both *HiTi* and *NR* as shown in Fig 16. Although the search space and searching time of *PA* at the clients are greater than that of *CH*, the total energy cost of *PA* can be lower since the tune-in cost of *CH* is prohibitively large as shown in Fig. 15.

Fig. 19 summarizes the number of settled vertices and relaxed edges under different settings at the traffic server for generating the broadcast index. As shown in Fig. 19a and b, both numbers of settled vertices and relaxed edges of *PA* are much smaller than that of *NR*. This is a bit surprise since *PA* needs additional cost for constructing and maintaining the different levels of *SN*. On average, *PA* settles two orders of magnitude fewer vertices, and relaxed one order of magnitude fewer edges, compared with *NR*. Note that in the last two settings, the time to collect the statistics for *NR* exceeds 15 h, and thus be omitted. Recall the cost analysis in Section 7.4, we confirm in here that $Dij(MLSN(R_i, R_j))$ is much smaller than $Dij(G)$. Even with two additional terms (in Section 7.4), *PA* still beats *NR* in terms of index re-generation cost. However, compared with *HiTi*, both *PA* and *NR* require more index re-generation cost because both of them need to perform pre-processing to reduce the set of regions for a client to build its local road network RN_C. This is the tradeoff for smaller size of RN_C at clients achieved from *PA*. *CH* has a slightly larger search space than *HiTi* among all settings.

It should be noted that we use the classic Dijkstra algorithm in our cost analysis and experimental studies for easy comparisons. More advanced algorithms [9,2] can

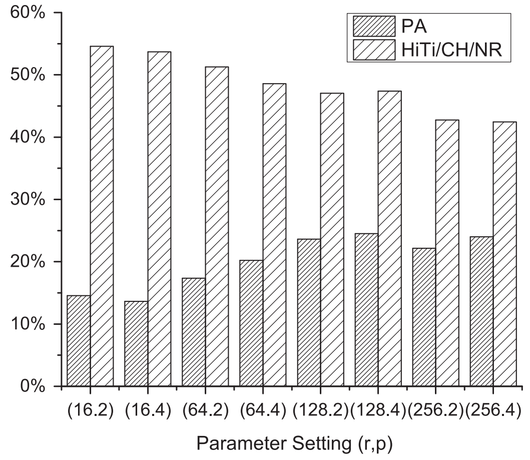


Fig. 22. Percentage of path changes (# path changes/# invocations).

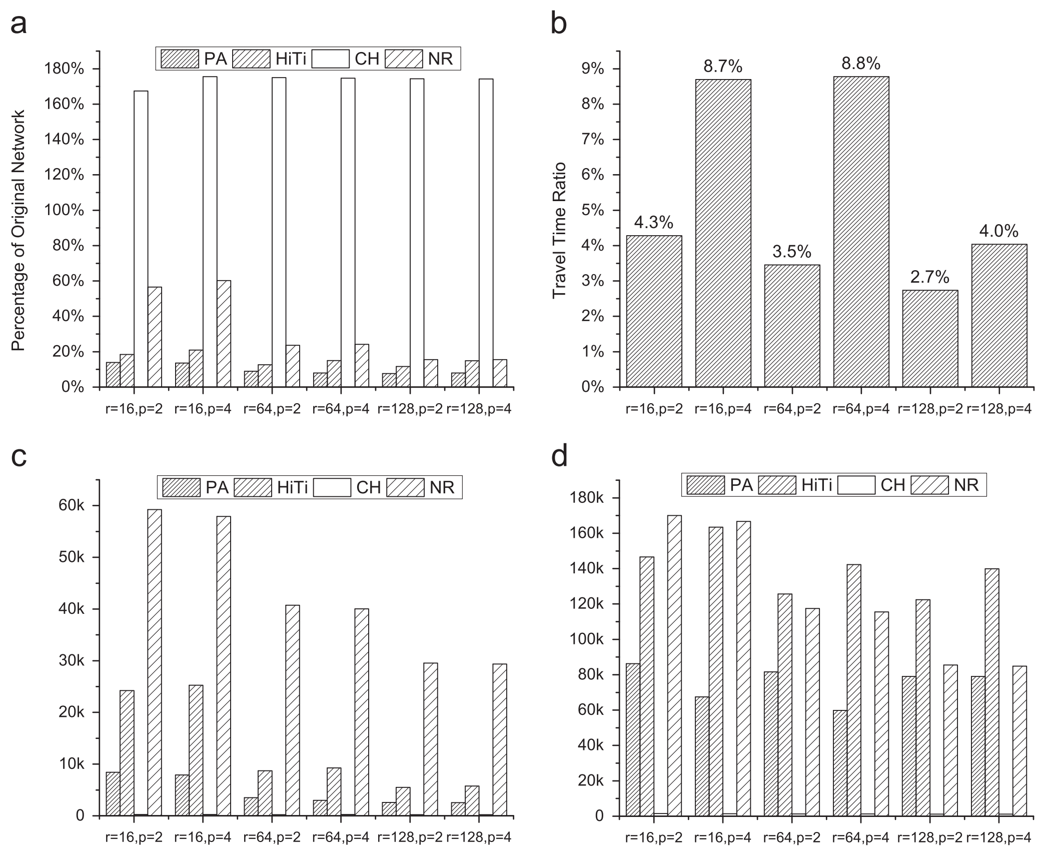


Fig. 23. The performance measures for data set NY. (The experiments for NY are obtained over $Q=100$ random queries with distance larger than $d=500{,}000$. Each time the client reaches a new region, $N=50{,}000$ updates are generated with update range $I=[0, 500]$.) (a) Size of client road network RN_C. (b) Travel time to destination. (c) Search space at clients (# of settled vertices). (d) Search space at clients (# of relaxed edges).

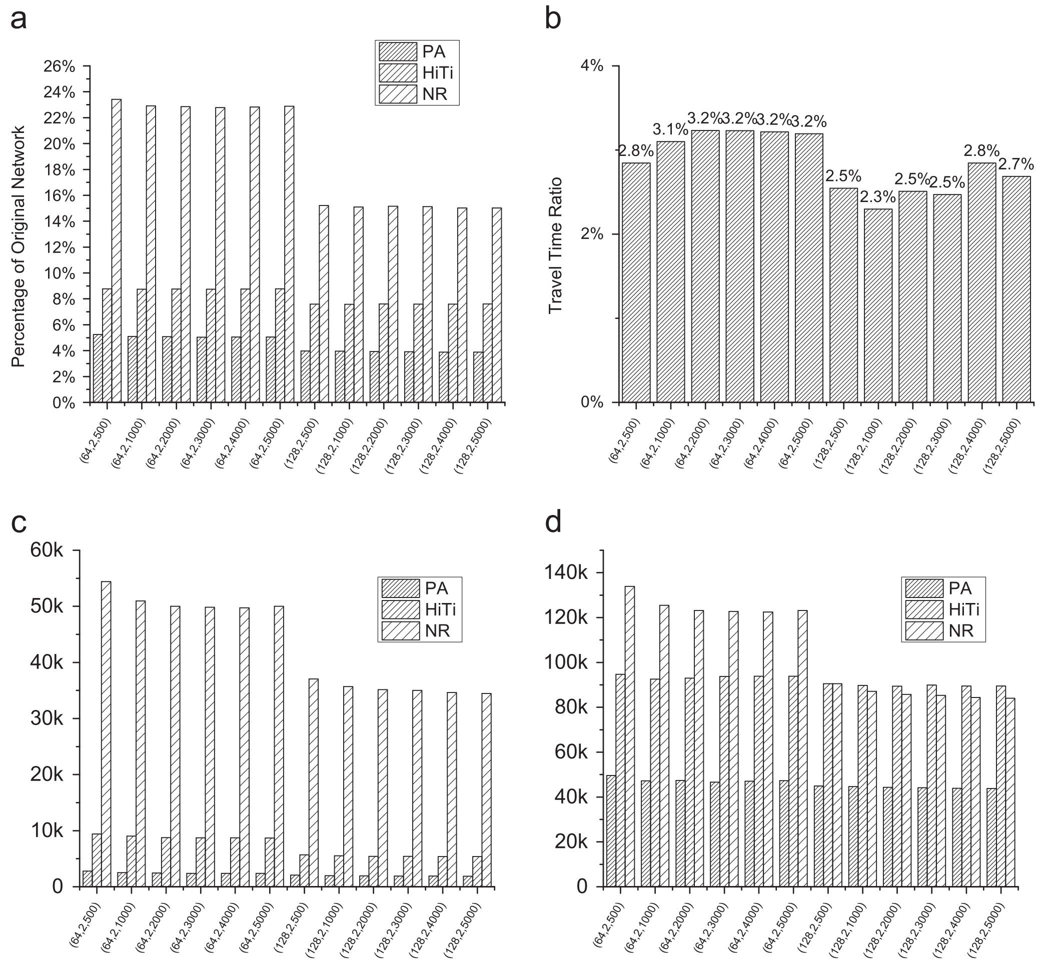


Fig. 24. The performance measures for more updates in BAY. (a) Size of client road network RN_C. (b) Travel time to destination. (c) # of settled vertices. (d) # of relaxed edges.

speedup the Dijkstra algorithm by three orders of magnitude in terms of search time, by including a pre-processing procedure. Even taking the cost of pre-processing into consideration, the total cost of the shortest path search can be greatly smaller than that of the Dijkstra algorithm. Another important observation is that the computation in *PA* includes lots of many-to-many shortest path searches for which there should have more efficient algorithms [18]. We leave the investigations of more efficient implementations at the server as an important future work.

In Fig. 20, we give the broadcast cycle size of the four methods (*PA*, *HiTi*, *CH* and *NR*). Each broadcast cycle consists of two parts: the broadcast index and network data. Each column in Fig. 20 corresponds to the total size of one broadcast cycle. As shown in Fig. 20, the size of network data of *PA* is larger than that of *NR* as it contains different levels of *SN* while *NR* only contains the original road network. However, the increase in the network data is not great, at most 2 times of the road network in the range we tested. The size of network data of *PA* is similar to that of *HiTi* as both of them use hierarchical index. As can be observed in Fig. 20, it is interesting to see that the index size of *HiTi* is larger than that of *PA* under all settings, which implies $Req_{HiTi} > Req_{PA}(G)$ in the cost analysis (Section 7.4). This reflects that the pruning mechanism and approximation technique in *PA* can effectively reduce the number of *SN* in $MLSN_C$. Consistent with our expectation, $Req_{NR}(G)$ is greater than $Req_{PA}(G)$. By using *SN* of higher level regions in $MLSN_C$ instead of the lowest level regions, the index size of *PA* could be greatly smaller than that of *NR*. This is especially obvious when *r* is large. When $r=256$, the increase in index data in *NR* makes it comparable to the size of network data, leading to a larger broadcast cycle size compared with *PA*, as shown in Fig. 20. For *CH*, the broadcast cycle size remains similar under different settings, because different values for *r* and *p* only affect when the *SPCQ* is invoked instead of the cycle size.

As shown in Fig. 21, by using *PA*, the travel time is slightly longer than *NR*, *HiTi* and *CH* by 5% on average. It is because *PA* only computes the shortest path containing different levels of shortcuts to the destination while *NR*, *HiTi* and *CH* calculate the exact path to the destination, and therefore, they have the same travel time. The slight increase in travel time in *PA* is the trading cost for the great reduction in tune-in and path searching costs at the

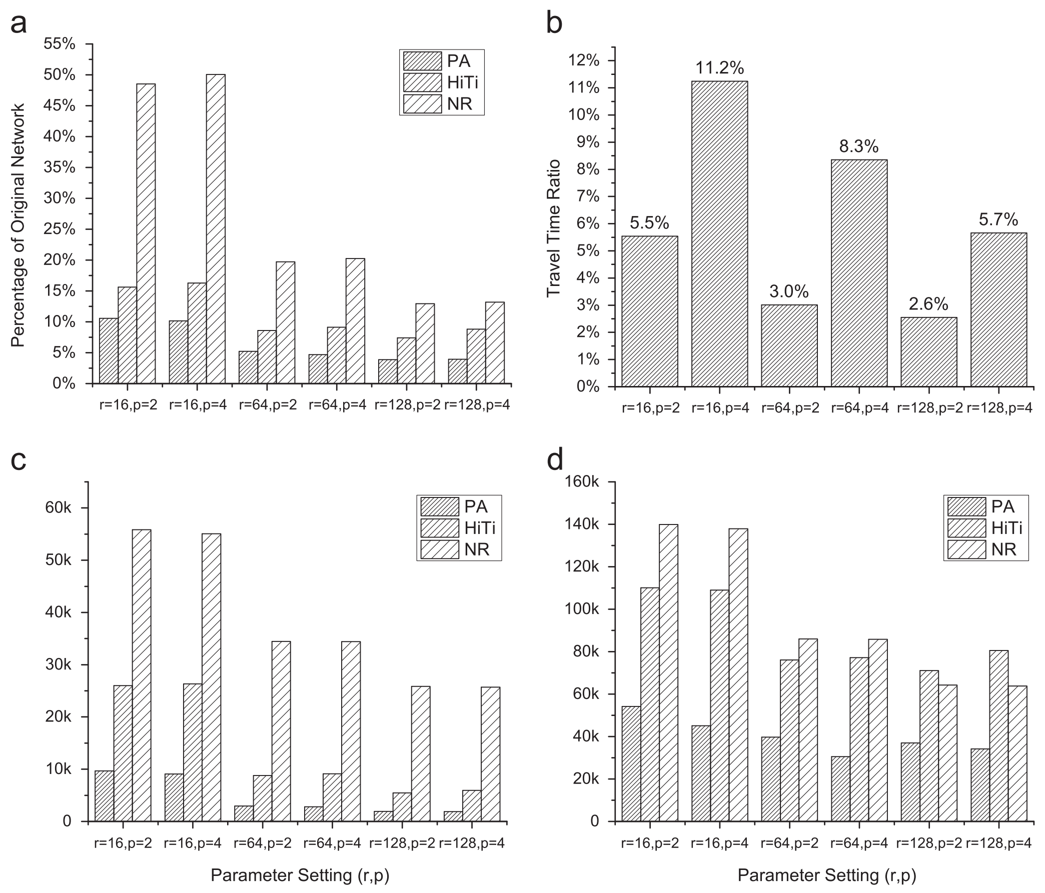


Fig. 25. The performance measures for shorter path in BAY. (a) Size of client road network RN_C. (b) Travel time to destination. (c) Search space at clients (# of settled vertices). (d) Search space at clients (# of relaxed edges).

clients. Another important observation is that the travel time of *PA* is smaller when $p=2$ than $p=4$. It is because the finer grouping makes the path to the destination region more accurate. However, the expense is larger network data as can be seen in Fig. 20.

To study the effectiveness of the shortest path search, we also measured the average number of path changes in an *SPCQ* for *PA*, *NR*, *HiTi* and *CH* under the same traffic updates. As can be seen in Fig. 22, on average, only 20% of the invocations of *PA* will have a different path compared with the path obtained in the previous invocation, while 50% of *NR*'s, *HiTi*'s or *CH*'s invocations give a different path. This is consistent with our expectation as *PA* uses higher level shortcuts and ignores lower level detail changes. The greater number of path changes confirms our belief that using higher level shortcut networks for regions farther away from the current region of the client can effectively monitor the shortest path. *NR*, *HiTi* and *CH* have similar performance in number of path changes since all of them calculate the exact path to destination.

When the underlying data set is NY, the relative performance of the four methods remains similar to that of BAY. We show the size of RN_C of some selected settings in Fig. 23a. Consistent with BAY, the size of client road network, RN_C of *PA* is significantly smaller than that of *NR*, *HiTi* and *CH*. When the number of regions increases from 16 to 128, smaller size of RN_C is received by the clients for all methods except *CH*. The travel time of *PA* is also not affected seriously as shown in Fig. 23b, with at most 8.8% larger than that of *NR* and *HiTi*.

Since the tune-in cost of *CH*, i.e., the size of client road network RN_C, is much greater than the other three methods, in the remaining experiments, we only compare the performances of *PA*, *HiTi* and *NR*.

8.2. Effects of update range

In this set of experiment, more updates are generated in each broadcast cycle ($N = 1,000,000$) and different values of update range (e.g., $I = 500, 1000, 2000, 3000, 4000, 5000$) are used to test their effects on the performances of *PA*, *HiTi* and *NR*. As shown in Fig. 24, consistent with the results presented in Section 8.1, the size of client road network and the number of vertices and edges in it under *PA* are significantly lower than that under *HiTi* and *NR*. The absolute values of the metric are similar for different values of range of update and the delay in arrival time in *PA* is smaller, e.g., around 2.5%, when the number of regions is greater.

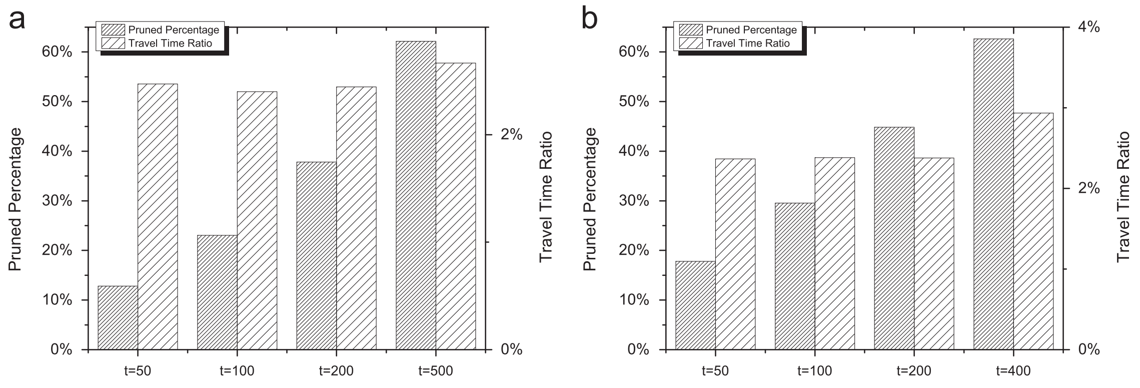


Fig. 26. Update pruned percentage and travel time for $r = 64$ (a) and 128 (b); $p = 2$.

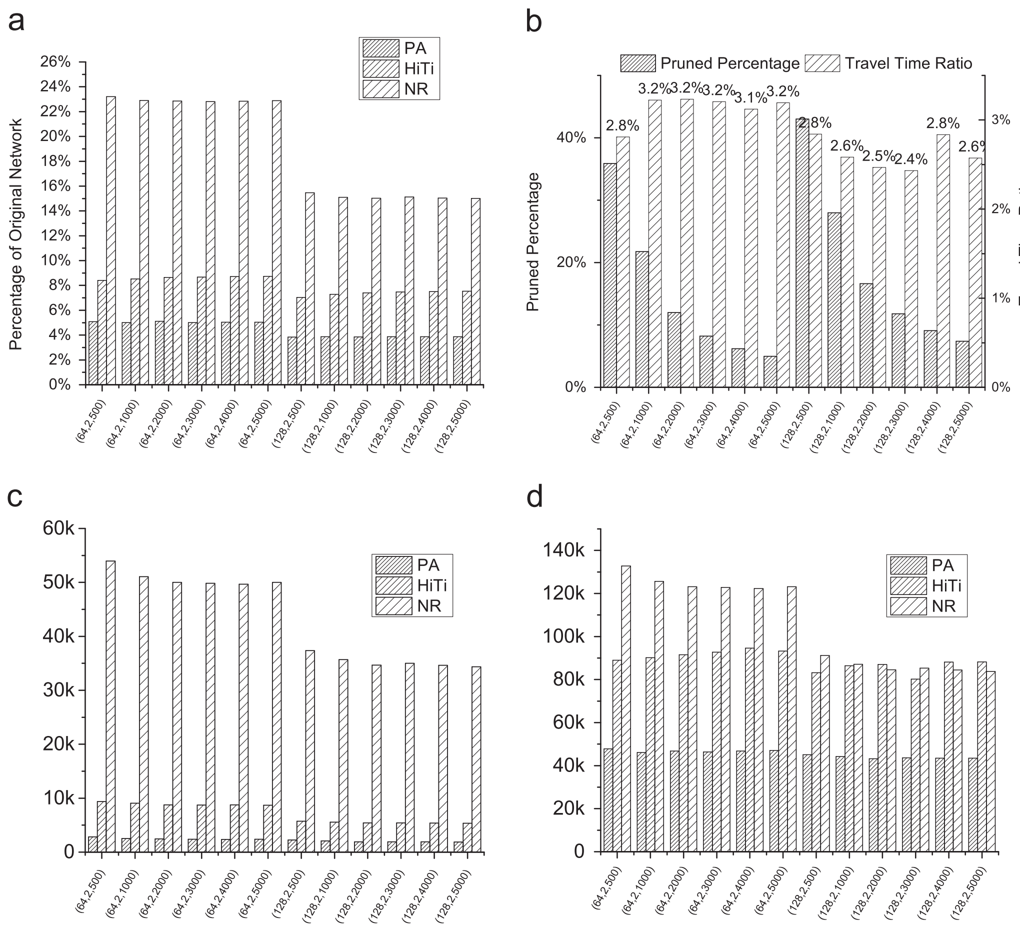


Fig. 27. The performance measures for more updates in BAY with update prunes ($t = 200$). (a) Size of client road network RN_C. (b) Update pruned percentage and travel time. (c) # of settled vertices. (d) # of relaxed edges.

8.3. *Effects of shorter path*

Fig. 25 shows the performance results of the three methods when the value for d is set to be 200,000, i.e., the distance between the start point and destination has to be equal to or greater than 200,000. Consistent with previous performance results, *PA* gives the smallest size of RN_C by a client for the shortest path search compared with both *NR* and *HiTi*. The tradeoff is longer travel time to destination. We observe that the absolute values are quite similar to the previous results. This implies that the average size of network data required for an invocation is similar regardless of the query distance. The index re-generation cost of *PA* is still lower than that of *NR* and is greater than that of *HiTi*.

8.4. Effects of update threshold

To study the effects of update threshold on the performance of *PA*, different values of the update thresholds were tested. The pruning ability was measured by the number of traffic updates that were pruned without affecting the shortest path, divided by the total number of updates that change the weights.

We conducted the experiments by varying the values of t, while fixing $r=64$, $p=2$, $I=[0, 1000]$, $N=100{,}000$ and $m=2$. Fig. 26 shows the results when different update thresholds were used. As expected, a loose threshold of $t=500$ can prune a large portion (60%) of the updates, while a tighter threshold $t=50$ and $t=100$ only prune 20% and 30%'s updates, respectively. It is interesting to see that except for very large values, the travel times to destinations remain similar even after pruning the updates using different values of update threshold. Thus, we may choose a loose threshold, e.g., $t=200$ for a lower index re-generation cost without seriously affecting the travel times to destinations. The experiments were repeated for $r=128$ while other settings remained the same. The results are referred to Fig. 26b. Consistent with previous settings, a loose threshold $t=200$ can prune large portion of updates, while the travel times are not affected by strong pruning.

Fig. 27 shows the results when the update range is varied and $t=200$. Similar to the results shown in Fig. 24, the tune-in and path searching costs at the clients are lower in *PA* compared with that in *HiTi* and *NR*. Compared with the results in Fig. 24, the absolute values of each metric only differ slightly when t is set to be 200. Therefore, we may impose the update prune threshold for lower index generation cost. Another observation is that the percentages of update pruned decrease with an increase of update range as shown in Fig. 27b. It is because with a larger value of I, the update threshold induced by $t=200$ becomes relatively smaller and can only prune a smaller portion of updates. Therefore, we may set a larger t to prune more updates when update range I is larger to aim for a lower index generation cost.

Summary of experimental results. As a summary, *PA* performs consistently better than *HiTi* and *NR* both in terms of tune-in cost and path searching cost. *NR* is the worst in terms of the costs both at the clients and server. Although the path searching cost of *CH* is the lowest amongst the four methods, the size of the client road network and tune-in cost of *CH* is much larger than that of *PA*, *HiTi* and *NR*. For *HiTi*, the reduction in the number of vertices is greater than the reduction in the number of edges, making the path searching cost at the clients expensive. In some settings (Fig. 18), the number of relaxed edges in *HiTi* is even larger than that of *NR*. The index generation cost of *PA* is significantly lower than that of *NR* while the index generation costs of *HiTi* and *CH* are lower. Therefore, as a conclusion, *PA* is more suitable than the other three methods for the clients where minimizing the tune-in and path searching costs are important, e.g., the clients have limited processing power and energy supply.

9. Conclusions and future work

In this paper, we study the shortest path searching problem at mobile clients using data broadcast to distribute traffic information to mobile clients for execution of *shortest path continuous queries* (*SPCQ*). Since the traffic data are changing, periodic execution of *SPCQ* is required to be performed at the clients, resulting in a heavy path searching cost. To resolve the problem, we propose the *progressive approach* (*PA*) with the purpose of minimizing both the tune-in and path searching costs at the clients. In *PA*, the path information to be distributed to the clients consists of road segments of the current regions of the clients and shortcuts from different levels of *shortcut networks (SN)*. A higher level *SN* will be chosen for a region if it is farther away from the current region of the client. Since the local network constructed from broadcast data by a client is a simplified hierarchical network with smaller search space, both the tune-in cost and path searching cost for the shortest path can be greatly reduced. Extensive experiments have been conducted to show the effectiveness of *PA* in reducing the costs with just small increase in arrival times to destinations as compared with *NR*, *HiTi* and *CH*. The performance results show that *PA* gives better performance than *NR* at both the traffic server and clients. Although the index generation cost at the traffic server is higher in *PA* comparing with *HiTi*, it gives lower tune-in and path searching costs at the clients to make *PA* suitable to the clients which have limited processing power and energy supply. An important future work is to find an optimal organization to merge the regions into the region tree to maximize the system performance. Finally, another important future work is to study how to reduce the index generation cost at the server by adopting a more efficient path searching algorithm.

References

[1] S. Acharya, R. Alonso, M. Franklin, S. Zdonik, Broadcast disks: data management for asymmetric communications environments, in: Proceedings of the ACM SIGMOD Conference, 1995, pp. 199–210.

[2] J. Arz, D. Luxen, P. Sanders, Transit node routing reconsidered, in: Proceedings of International Symposium on Experimental Algorithms, 2013, pp. 55–66.

[3] G. Batz, R. Geisberger, P. Sanders, C. Vetter, Minimum time-dependent travel times with contraction hierarchies, ACM J. Exp. Algorithm. 18 (1.4) (2013).

[4] Y. Cai, A. Hua, G. Gao, Processing range-monitoring queries on heterogeneous mobile objects, in: Proceedings of IEEE International Conference on Mobile Data Management, 2004, pp. 27–38.

[5] J. Chen, R. Cheng, Efficient evaluation of imprecise location-dependent queries, in: Proceedings of IEEE ICDE Conference, 2007, pp. 586–595.

[6] R. Cheng, J. Chen, M. Mokbel, C. Chow, Probabilistic verifiers: evaluating constrained nearest-neighbor queries over uncertain data, in: Proceedings of IEEE ICDE Conference, 2008, pp. 973–982.

[7] R. Cheng, B. Kao, A. Kwan, S. Prabhakar, Y. Tu, Filtering data streams for entity-based continuous queries, IEEE Trans. Knowl. Data Eng. 22 (2) (2010) 234–248.

[8] Y.D. Chung, S. Yoo, M.H. Kim, Energy- and latency-efficient processing of full-text searches on a wireless broadcast stream, IEEE Trans. Knowl. Data Eng. 22 (2) (2010) 207–218.

[9] D. Delling, A. Goldberg, A. Nowatzyk, R. Werneck, PHAST: hardware-accelerated shortest path trees, J. Parallel Distrib. Comput. 73 (7) (2013) 940–952.
[10] B. Gedik, L. Liu, Distributed processing of continuously moving queries on moving objects in a mobile system, in: Proceedings of Extending Database Technology, 2004, pp. 67–87.
[11] R. Geisberger, P. Sanders, D. Schultes, D. Delling, Contraction hierarchies: faster and simpler hierarchical routing in road networks, in: Proceedings of International Workshop on Experimental Algorithms, 2008, pp. 319–333.
[12] R. Geisberger, P. Sanders, D. Schultes, C. Vetter, Exact routing in large road networks using contraction hierarchies, Transp. Sci. 46 (3) (2012) 388–404.
[13] T. Imielinski, S. Viswanathan, B. Badrinath, Power efficient filtering of data on air, in: Proceedings of Extending Database Technology, 1994, pp. 245–258.
[14] T. Imielinski, S. Viswanathan, B. Badrinath, Data on air: organization and access, IEEE Trans. Knowl. Data Eng. 9 (3) (1997) 353–372.
[15] Y. Jing, C. Chen, W. Sun, B. Zheng, L. Liu, C. Tu, Energy-efficient shortest path query processing on air, in: Proceedings of the ACM GIS Conference, 2011, pp. 393–396.
[16] S. Jung, S. Pramanik, An efficient path computation model for hierarchically structured topographical road maps, IEEE Trans. Knowl. Data Eng. 14 (5) (2002) 1029–1046.
[17] G. Kellaris, K. Mouratidis, Shortest path computation on air indexes, Proc. VLDB 3 (1) (2010) 747–757.
[18] S. Knopp, P. Sanders, D. Schultes, F. Schulz, D. Wagner, Computing many-to-many shortest paths using highway hierarchies, in: Proceedings of Workshop on Algorithm Enginerring and Experiments, 2007.
[19] K.-Y. Lam, E. Chan, H. Leung, M. Au, Concurrency control strategies for ordered data broadcast in mobile computing systems, Inf. Syst. 29 (3) (2004) 207–234.
[20] Z. Li, K. Lee, B. Zheng, W.-C. Lee, D. Lee, X. Wang, IR-tree: an efficient index for geographic document search, IEEE Trans. Knowl. Data Eng. 23 (4) (2011) 585–599.
[21] M. Mokbel, X. Xiong, W. Aref, SINA: scalable incremental processing of continuous queries in spatio-temporal databases, in: Proceedings of ACM SIGMOD Conference, 2004, pp. 623–634.
[22] K. Mouratidis, S. Bakiras, D. Papadias, Continuous monitoring of spatial queries in wireless broadcast environments, IEEE Trans. Mob. Comput. 8 (10) (2009) 1297–1311.
[23] S. Nutanong, E. Tanin, J. Shao, R. Zhang, K. Ramamohanarao, Continuous detour queries in spatial networks, IEEE Trans. Knowl. Data Eng. 24 (7) (2012) 1201–1215.
[24] C.K. Poon, C.J. Zhu, Energy-efficient air-indices for distance queries on road networks, in: Proceedings of the ACM GIS Conference, 2012, pp. 558–561.
[25] M. Qiao, H. Cheng, L. Chang, J. Yu, Approximate shortest distance computing: a query-dependent local landmark scheme, in: Proceedings of IEEE ICDE Conference, 2008, pp. 462–473.
[26] Y. Sasaki, W.-C. Lee, T. Hara, S. Nishio, On alleviating beacon overhead in routing protocols for urban VANETs, in: Proceedings of IEEE International Conference on Mobile Data Management, 2013, pp. 66–76.
[27] W. Sun, Y. Qin, J. Wu, B. Zheng, Z. Zhang, P. Yu, P. Liu, J. Zhang, Air indexing for on-demand XML data broadcast, IEEE Trans. Knowl. Data Eng. 25 (6) (2014) 1371–1381.
[28] L.H. U, H.J. Zhao, M.L. Yiu, Y. Li, Z. Gong, Towards online shortest path computation, IEEE Trans. Knowl. Data Eng. 26 (4) (2014) 1012–1025.
[29] A.B. Waluyo, D. Taniar, W. Rahayu, B. Srinivasan, Clustering-based index and data broadcasting for mobile nearest neighbor query processing, IEEE Trans. Knowl. Data Eng. 9 (4) (2013) 1964–1974.
[30] J. Xu, W.-C. Lee, X. Tang, Q. Gao, S. Li, An error-resilient and tunable distributed indexing scheme for wireless data broadcast, IEEE Trans. Knowl. Data Eng. 18 (3) (2006) 392–404.
[31] C. Young, G. Chiu, Efficient dissemination of transaction-consistent data in broadcast environments, IEEE Trans. Knowl. Data Eng. 19 (3) (2007) 384–397.
[32] B. Zheng, W.-C. Lee, D. Lee, Spatial queries in wireless broadcast systems, Wirel. Netw. 10 (6) (2004) 723–736.
[33] B. Zheng, W.-C. Lee, K. Lee, D. Lee, M. Shao, A distributed spatial index for error-prine wireless data broadcast, VLDB J. 18 (4) (2009) 959–986.
[34] B. Zheng, W.-C. Lee, P. Liu, D. Lee, X. Ding., Tuning on-air signatures for balancing performance and confidentiality, IEEE Trans. Knowl. Data Eng. 21 (12) (2009) 1783–1797.