ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

Online Shortest Path Computation Based on Live Traffic Index

Ch.Venkateswarlu, B.V.Sri Ram, V.Sindhu

Assistant Professor, Dept. of MCA, Narayana Engineering College, Nellore, AP, India

Student, Dept. of MCA, Narayana Engineering College, Nellore, AP, India

Student, Dept. of MCA, Narayana Engineering College, Nellore, AP, India

ABSTRACT: The online shortest path problem aims at computing the shortest path based on live traffic circumstances. This is very important in modern car navigation systems as it helps drivers to make sensible decisions. To our best knowledge, there is no efficient system/solution that can offer affordable costs at both client and server sides for online shortest path computation. Unfortunately, the conventional client-server architecture scales poorly with the number of clients. A promising approach is to let the server collect live traffic information and then broadcast them over radio or wireless network. This approach has excellent scalability with the number of clients. Thus, we develop a new framework called live traffic index (LTI) which enables drivers to quickly and effectively collect the live traffic information on the broadcasting channel. An impressive result is that the driver can compute/update their shortest path result by receiving only a small fraction of the index. Our experimental study shows that LTI is robust to various parameters and it offers relatively short tune-in cost (at client side), fast query response time (at client side), small broadcast size (at server side), and light maintenance time (at server side) for online shortest path problem.

KEYWORDS: Shortest path, air index, broadcasting

I. INTRODUCTION

SHORTEST path computation is an important function in modern car navigation systems and has been extensively studied in [1], [2], [3], [4], [5], [6], [7], [8]. This function helps a driver to figure out the best route from his current position to destination. Typically, the shortest path is computed by offline data pre-stored in the navigation systems and the weight (travel time) of the road edges is estimated by the road distance or historical data. Unfortunately, road traffic circumstances change over time. Without live traffic circumstances, the route returned by the navigation system is no longer guaranteed an accurate result. We demonstrate this by an example in Fig. 1. Suppose that we are driving from Lord & Taylor (label A) to Mt Vernon Hotel Museum (label B) in Manhattan, NY. Those old navigation systems would suggest a route based on the pre-stored distance information as shown in Fig. 1a. Note that this route passes through four road maintenance operations (indicated by maintenance icons) and one traffic congested road (indicated by a red line). In fact, if we take traffic circumstances into account, then we prefer the route in Fig. 1b rather than the route in Fig. 1a. Nowadays, several online services provide live traffic data (by analyzing collected data from road sensors, traffic cameras, and crowd sourcing techniques), such as Google- Map [9], Navteq [10], INRIX Traffic Information Provider [11], and Tom Tom NV [12], etc. These systems can calculate the snapshot shortest path queries based on current live traffic data; however, they do not report routes to drivers continuously due to high operating costs. Answering the shortest paths on the live traffic data can be viewed as a continuous monitoring problem in spatial databases, which is termed online shortest paths computation (OSP) in this work. To the best of our knowledge, this problem has not received much attention and the costs of answering such continuous queries vary hugely in different system architectures. Typical client-server architecture can be used to answer shortest path queries on live traffic data. In this case, the navigation system typically sends the shortest path query to the service provider and waits the result back from the provider (called result transmission model). However, given the rapid growth of mobile devices and services, this model is facing scalability limitations in terms of network bandwidth and server loading. According to the Cisco Visual Networking Index forecast [13], global mobile traffic in 2010 was 237 petabytes per month and it grew by 2.6-fold in 2010, nearly tripling for the third year in a row. Based on a telecommunication expert [14], the world's

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

cellular networks need to provide 100 times the capacity in 2015 when compared to the networks in 2011. Furthermore, live traffic are updated frequently as these data can be collected by using crowd sourcing techniques (e.g., anonymous traffic data from Google map users on certain mobile devices). As such, huge communication cost will be spent on sending result paths on the this model. Obviously, the client-server architecture will soon become impractical in dealing with massive live traffic in near future. Ku et al. [15] raise the same concern in their work which processes spatial queries in wireless broadcast environments based on Euclidean distance metric. Malviya et al. [16] developed a client-server system for continuous monitoring of registered shortest path queries. For each registered query ðs; tÞ, the server first pre computes K different candidate paths from s to t. Then, the server periodically updates the travel times on these K paths based on the latest traffic, and reports the current best path to the corresponding user. Since this system adopts the _ L.H. U, H.J. Zhao, Y.H. Li, and Z. Gong are with the Department of Computer and Information Science, University of Macau, Av. Padre Tom_as Pereira, Taipa, Macau, China. client-server architecture, it cannot scale well with a large number of users, as discussed above. In addition, the reported paths are approximate results and the system does not provide any accuracy guarantee.

II. BACKGROUND OF RELATED WORK

In this section, we briefly discuss the applicability of the state-of-the-art shortest path solutions on different transmission models. As discussed in the introduction, the result transmission model scales poorly with respect to the number of clients. The communication cost is proportional to the number of clients (regardless of whether the server transmits live traffic or result paths to the clients). Thus, we omit this model from the remaining discussion. 2.2.1 Raw Transmission Model Under the raw transmission model, the traffic data (i.e., edge weights) are broadcasted by a set of packets for each broadcast cycle. Each header stores the latest time stamp of the packets, so that clients can decide which packets have been updated, and only fetch those updated packets in the current broadcast cycle. Having downloaded the raw traffic data from the broadcast channel, the following methods either directly calculate the shortest path or efficiently maintain certain data structure for the shortest path computation. Uninformed search (e.g., Dijkstra’s algorithm) traverses graph nodes in ascending order of their distances from the source s, and eventually discovers the shortest path to the destination t. Bi-directional search (BD) [3] reduces the search space by executing Dijkstra’s algorithm simultaneously forwards from s and backwards from t. As to be discussed shortly, bi-directional search can also be applied on some advanced index structures. However, the response time is relatively high and the clients may receive large amount of irrelevant updates due to the transmission model. Goal directed approaches search towards the target by filtering out the edges that cannot possibly belong to the shortest path. The filtering procedure requires some pre-computed information. ALT [25] and arc flags (AF) [26] are two representative algorithms in this category. ALT makes use of A_ search, landmarks, and triangle inequality [27]. A few landmark nodes are selected and the distances between each landmark and every node are pre-computed. These pre-computed distances can be exploited to derive distance bounds for A_ search on the graph. Delling and Wagner [28] proposes a lazy update paradigm for ALT (DALT) so that it can tolerate certain extents of edge weights changes on a dynamic graph. The distance bounds derived from the pre-computed information remain correct if no edge weight becomes lower than the initial weight used at the ALT construction. This lazy update paradigm significantly reduces the index maintenance cost. Another well known goal directed approach is arc flags that partitions the graph into m sub-graphs. For each edge e, it stores a bitmap B where B½i_ is set to true if and only if a shortest path to a node in the sub-graph i starts with e. During the Dijkstra execution, it only relaxes those edges for which the bitmap flag of the target node’s subgraph is true. AF provides reasonable speed-ups, but consume too much space for large road networks. The dynamic updates of AF (DAF) has been recently studied in [29]. However, the solution is not practical since the cost of updating the bitmap flags is exponential to the number of edge updates. Dynamic shortest path tree (DSPT) maintains a tree structure locally for efficient shortest path retrieval. Chan and Yang [22] discusses how to maintain a correct shortest path tree rooted at s after receive a set of edge weight updates to the graph G ¼ ðV ;EÞ. Finding a shortest path from s to any node is computed at OðjV jÞ time on the shortest path tree. In their work, a simple dynamic version of Dijkstra is proposed which can outperform all competitors. 2.2.2 Index Transmission Model The index transmission model enables servers to broadcast an index instead of raw traffic data. We review the state-ofthe- art indices for shortest path computation and discuss their applicability on the index transmission model. Road map hierarchical approaches try to exploit the hierarchical structure to the road map network in a pre-processing step, which can be used to accelerate all subsequent queries. These speed-up approaches include reach [4], highway hierarchies (HH) [2][6], contraction hierarchies (CH)

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

[30], and transitnode routing (TNR) [1]. Reach, HH, and CH are based on shortcut techniques [2][6], i.e., some paths in the original graph are represented by some shortcut edges. The shortcuts are identified out by exploiting the hierarchical structure (e.g., node ordering) on the road map network. To answer a query, a bi-directional search is executed on the overlay graph that constitutes of the shortcuts and some edges in the original graph. As the shortcuts are the only extra structure stored in the index, the construction is relatively fast as compared to other index approaches. TNR is based on a simple observation that a driving path only passes one of a few important transit nodes. However, the maintenance time could be high as most of them have no efficient approach to update the pre-computed data structure. HH and CH can support dynamic weight updates [7] but the solution is limited to weight increasing cases. In [32], a theoretical approach has been proposed to update the overlay graphs, but the proposed algorithms have not been shown to have good practical performances in real-world networks. Again, none of these approaches supports index transmission model well since the shortest path can only be computed on a complete index.Hierarchical index structures provide another way to abstracting and structuring a topographical index in a hierarchical fashion. Hierarchical MulTi-graph model (HiTi) [21] is a representative approach in this category. The meaning of hierarchy in HiTi is the hierarchy of the index (i.e., tree structure) instead of the hierarchy of the road map (i.e., level of roads). By exploiting the hierarchical index structure, HiTi can support fast shortest path computation on a portion of entire index which can significantly reduce the tune-in cost on the index transmission model. However, prohibitive maintenance time and large broadcast size make it inapplicable to OSP on any transmission model. Hierarchical encoded path view (HEPV) [20] and Hub indexing [19] share the same intuition of HiTi which divides large graph into smaller sub graphs and organize them in ahierarchical fashion by pushing up border nodes. However, both are infeasible for OSP since these approaches suffer from the excessive storage overhead for a large amount of pre-computed path information. TEDI [33] applies a tree-based partitioning on the graph Such that each partition in the tree has a bounded number of sub-partitions.

III. CONTRIBUTION

An alternative solution is to broadcast live traffic data over wireless network (e.g., 3G, LTE, Mobile WiMAX, etc.). The navigation system receives the live traffic data from the broadcast channel and executes the computation locally (called raw transmission model). The traffic data are broadcasted by a sequence of packets for each broadcast cycle. To answer shortest path queries based on live traffic circumstances, the navigation system must fetch those updated packets for each broadcast cycle. However, as we will analyze an example in Section 2.2, the probability of a packet being affected by 1% edge updates is 98.77%. This means that clients almost fetch all broadcast packets in a broadcast cycle. The main challenge on answering live shortest paths is scalability, in terms of the number of clients and the amount of live traffic updates. A new and promising solution to the shortest path computation is to broadcast an air index over the wireless network (called index transmission model) [17], [18]. The main advantages of this model are that the network overhead is independent of the number of clients and every client only downloads a portion of the entire road map according to the index information. For instance, the proposed index in [17] constitutes a set of pair wise minimum and maximum traveling costs between every two sub partitions of the road map. However, these methods only solve the scalability issue for the number of clients but not for the amount of live traffic updates. As reported in [17], the re-computation time of the index takes 2 hours for the San Francisco (CA) road map. It is prohibitively expensive to update the index for OSP, in order to keep up with live traffic circumstances. Motivated by the lack of off-the-shelf solution for OSP, in this paper we present a new solution based on the index transmission model by introducing live traffic index (LTI) as the core technique. LTI is expected to provide relatively short tune-in cost (at client side), fast query response time (at client side), small broadcast size (at server side), and light maintenance time (at server side) for OSP. We summarize LTI features as follows. _ The index structure of LTI is optimized by two novel techniques, graph partitioning and stochastic-based construction, after conducting a thorough analysis on the hierarchical index techniques [19], [20], [21]. To the best of our knowledge, this is the first work to give a thorough cost analysis on the hierarchical index techniques and apply stochastic process to optimize the index hierarchical structure. (Section 4) _ LTI efficiently maintains the index for live traffic circumstances by incorporating Dynamic Shortest Path Tree (DSPT) [22] into hierarchial index techniques. In addition, a bounded version of DSPT is proposed to further reduce the broadcast overhead. (Section 6) _ By incorporating the above features, LTI reduces the tune-in cost up to an order of magnitude as compared to the state-of-the-art competitors; while it still provides competitive query response time, broadcast size, and maintenance time. To the best of our knowledge, we are the first work that

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016

attempts to minimize all these performance factors for OSP.The rest of the paper is organized as follows. We first introduce four main performance factors for evaluating OSP and overview the state-of-the-art shortest path computation methods in Section 2.

IV. PROPOSED METHODOLOGY

In Section 4.1, we carefully analyze the hierarchical index structures and study how to optimize the index. In Section 4.2, we present a stochastic based index construction that minimizes not only the size overhead but also reduces the search space of shortest path queries. To the best of our knowledge, this is the first work to analyze the hierarchical index structures and exploit the stochastic process to optimize the index. 4.1 Analysis of Hierarchical Index Structures Hierarchical index structures (e.g., HiTi [21], HEPV [20], and Hub Indexing [19], TEDI [33]) enable fast shortest path computation on a portion of entire index which significantly reduces the tune-in cost on the index transmission model.Given a graph G ¼ ðVG; EGÞ (i.e., road network), this type of index structures partitions G into a set of small sub-graphs SGi and organizes SGi in a hierarchical fashion (i.e., tree). In Fig. 5, we illustrate a graph being partitioned into 10 subgraphs (SG1, SG2; . . . ; SG10) and the corresponding hierarchical index structure. Every leaf entry in a hierarchical structure represents a subgraph SGi that consists of the corresponding nodes and edges from the original graph. For instance, SG1 consists of two nodes VSG1 ¼ fa; bg and one edge ESG1 ¼ fða; bÞg. A non-leaf entry stores the inter-connectivity information between the child entries. For instance, SG1-2 stores a connectivity edge GSG1-2 ¼ fðb; cÞg between SG1 and SG2. To boost up the shortest path computation, the hierarchical index structures additionally keep some pre-computed information in the index entries. For instance, shortcuts DSGi are the most common type of pre-computed information in these indices, where a shortcut is the shortest path between two border nodes in a subgraph. In Fig. 5, SG5 has two border nodes2 k and m so that SG5 keeps a shortcut DSG5 ¼ fðk;mÞg and its corresponding weight.To answer a shortest path query qðs; tÞ using the hierarchical structures, a common approach is to fetch the relevant entries from the index using a bottom-up execution fashion. For the sake of analysis, we use HiTi as our reference model in the remaining discussion. Our analysis can be adapted to other approaches since their execution paradigm shares the same principle. In Fig. 5, the relevant entries of a shortest path query qðb; dÞ are shaded in gray color. Besides the source and destination leaf entries (SG1 and SG3), we need to fetch the entries from two leaf entries towards the root entry (SG1-2, SG1-3, SG1-5, and SG1-10) and their sibling entries (SG2, SG4-5, and SG6-10). The shortest path is computed on the search graph Gq (typically much smaller than G) which constitutes of the edges from the source and destination entries and the connectivity edges and shortcuts from other relevant entries. Note that the edges in Gq already secure the correctness of the shortest path query process [21]. As an example, suppose the shortest path of qðb; dÞ passes through an edge in SG6, this path must be revealed in the shortcut of SG6-10 (i.e., DSG6-10 ¼ fðf; pÞg). Cost analysis. The total space requirement of a hierarchical index I can be represented as follows. jIj ¼ X SGi2I ðjVSGij þ jESGij þ jGSGij þ jDSGi jÞ þ tree; (1) where VSGi and ESGi represent the nodes and edges in SGi, respectively, GSGi represents the connectivity information between the child entries, DSGi represents the pre-computed information kept in SGi, and tree represents the hierarchical information of I. Since VSGi , ESGi and GSGi are directly derived from the original graph and tree is negligible compared to G, the space requirement can be revised as the follows: jIj _ jGj þ X SGi2I jDSGi j: (2) To minimize the index broadcast size, it is more or less equivalent to minimize the size of DSGi . The simplest way is to partition the graph into multiple subgraphs such that the total size of DSGi is minimized. However, this may not optimize the query performance being discussed shortly. In our problem, both tune-in cost and query response time are highly relevant to the size of the search graph Gq (i.e., search space). Given an index I and a query q, the search space of q can be represented by the relevant edge sets: SðI; qÞ ¼ jESGs [ESGt [fGSGi [DSGi : 8SGi 2 Gq_stgj; (3) where SGs and SGt represent the leaf entry of source and destination, respectively and Gq_st ¼ Gq n fSGs [SGtg. To reduce SðI; qÞ for all possible queries, our goal is to find a hierarchical structure such that it minimizes (O1) the size of leaf entries ESGs and ESGt , (O2) the overhead of precomputed information DSGi, and (O3) the number of relevant entries Gq. However, these objectives are correlated to each other. For instance, to make ESGs and ESGt smaller, a simple way is to partition G into more subgraphs; however, it may increase the number of relevant entries Gq and the number of border nodes DSGi . 4.2 Index Construction The above discussion shows that it is hard to find a hierarchical index structure I that achieves all optimization objectives. One possible solution is to relax the optimization objectives which makes them be the tunable factors of the problem. While the overhead of pre-computed information (O2) and the number of relevant entries (O3) cannot be decided straightforwardly, we decide to relax the first objective (i.e., minimizing the size of leaf entries) such that it becomes a tunable factor in constructing the index. To minimize the overhead of pre computed information (O2), we study a graph partitioning optimization that minimizes the index overhead DSGi through the entire index construction subject to a leaf entry constraint (O1). Subsequently, we propose a stochastic process to optimize the index structure such that the size of the query search graph Gq is minimized (O3).Graph partitioning optimization. For the sake of discussion, we denote that the number of subgraphs being created is g that is a tunable parameter for controlling the number of subgraphs3 in this work. According to Eq. (2), minimizing the size of DSGi is likely to minimize the overhead of Obviously, our objective is to find a hierarchical index structure such that OBJ ðIÞ ¼ min SGi2I PjDSGi j minfjVSGi jg; (4) where minfjVSGi jg can be viewed as a normalized factor such that the objective function prefers balanced partitions. We observe that minimizing Eq. (4) is similar to finding the best Cheeger cut [40] in a graph. A Cheeger cut is to remove some edges from a graph such that the graph is isolated into n subgraphs subject to an objective function: OBJ CheegerðGÞ ¼ min SG1;...;SGn2G CutðfSG1; . . . ; SGngÞ minfjSG1j; . . . ; jSGnjg; (5) where CutðfSG1; . . . ; SGngÞ is the number of edges between any two subgraphs. We use an example to illustrate how Cheeger cut result can be viewed as a good result of I. Fig. 6a shows a Cheeger cut on a graph where the cut value CutðfSG1; SG2gÞ and the number of shortcut edges, jDSG1j þ jDSG2 j ¼ jf;gjþ jfðe; fÞ; ðe; gÞ; ðf; gÞgj, are identical (i.e., 3).

V. EXPERIMENTAL RESULTS

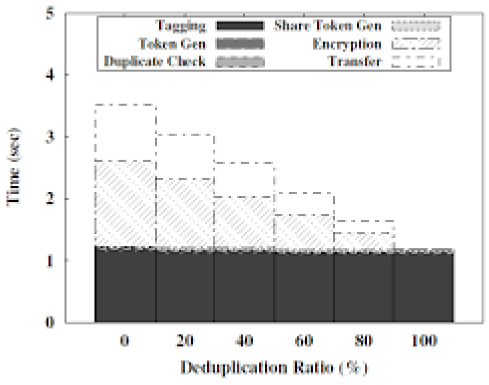
In this section, we empirically evaluate the performance of some representative algorithms using the broadcasting architecture; we ignore the client-server architecture due to massive live traffic in near future (see Section 1). From our discussion in Section 2, bi-directional search [3], ALT on dynamic graph (DALT) [28], and dynamic shortest paths tree [22], are applicable to raw transmission model. On the other hand, contraction hierarchies [30], Hierarchical MulTi-graph model [21], and our proposed live traffic index are applicable to index transmission model. We omit some methods (such as TNR [1], Quadtree [36], SHARC [39], and CALT [31]) due to their prohibitive maintenance time and broadcast size. In the following, we first describe the road map data used in experiments and describe the simulation of clients' movements and live traffic circumstances on a road map. Then, we study the performance of the above methods with respect to various factors. Map data. We test with four different road maps, including New York City (NYC) (264k nodes, 733k edges), San Francisco bay area road map (SF) (174k nodes, 443k edges), San Joaquin road map (SJ) (18k nodes, 48k edges), and Olden burg road map (OB) (6k nodes, 14k edges). All of them are available at [43] and [44]. Simulation of clients and traffic updates. We run the network kbased generator [44] to generate the weight of edges. It initializes 100,000 cars (i.e., clients) and then generates 1,000 new cars in each iteration. It runs for 200 iterations in total, with the other generator parameters as their default values. The weight of an edge is set to the average driving time on it. We adopt the approach in [28] to simulate live traffic updates. The initial weights of edges are assigned by the above network-based generator. In each iteration, we randomly select a set of edges subject to the update ratio d and specific weight update settings. In our work, each weight update can be either a light traffic change, a heavy traffic change, or a road maintenance. The proportion of these update types are b, 1_b 2 , and 1_b 2 , respectively, where U ET AL.: TOWARDS ONLINE SHORTEST PATH COMPUTATION 1021 b is a ratio parameter. For each light traffic change, the edge weight is set to 20% of the current weight. For each heavy traffic change, the weight is set to a large value by multiplying a weight factor v (which is set to 5 by default). For each road maintenance, the weight is set to 1. We reset the edge weight to its initial value if the edge weight is updated by heavy traffic or road maintenance after 10 iterations. Implementation and evaluation platforms. All tested methods except CH [30] were implemented in Java. Experiments on the service provider were conducted on an Intel Xeon E5620 2.40 GHz CPU machine with 18 GBytes memory, running Ubuntu 10.10; and experiments on the client were performed on an Intel Core2Duo 2.66 GHz CPU machine with 4GBytes memory, running Windows 7. Table 3 shows the ranges of the investigated parameters, and their default values (in bold). In each experiment, we vary a single parameter, while setting the

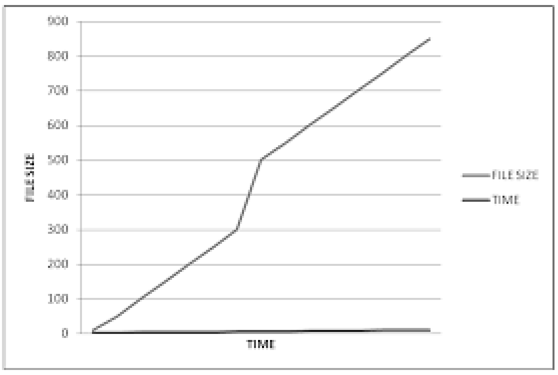
ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 6, June 2016





Acronym	Description
S-CSP	Storage-cloud service provider
POW	Proof of Ownership
(*PKU*,*skU*)	User’s public and secret key pair
KF	Convergent encryption key for file *F*
PU	Privilege set of a user *U*
PF	Specified privilege set of a file
F φ′F;p	Token of file *F* with privilege *p*

FIG:5.1 TABLE:SHORTEST PATH

others to their default values. For each method, we measure its performance in terms of tune-in size, query response time, broadcast size, and index maintenance time for all tested methods, and report its average performance over 2,000 shortest path queries.

VI. CONCLUSION

In this paper we studied online shortest path computation; the shortest path result is computed/updated based on the live traffic circumstances. We carefully analyze the existing work and discuss their inapplicability to the problem (due to their prohibitive maintenance time and large transmission overhead). To address the problem, we suggest a promising architecture that broadcasts the index on the air. We first identify an important feature of the hierarchical index structure which enables us to compute shortest path on a small portion of index. This important feature is thoroughly used in our solution, LTI. Our experiments confirm that LTI is a Pareto optimal solution in terms of four performance factors for online shortest path computation. In the future, we will extend our solution on time dependent networks. This is a very interesting topic since the decision of a shortest path depends not only on current traffic data but also based on the predicted traffic circumstances.

VII. ACKNOWLEDGEMENTS

This work was partially supported by SRG007-FST11-LHU and MYRG109(Y1-L3)-FST12-ULH from UMAC Research Committee and FDCT/106/2012/A3 from FDCT. Man Lung Yiu was supported by ICRG grant A-PL99 from the Hong Kong Polytechnic University.

REFERENCES

1. F. Wei, “TEDI: Efficient Shortest Path Query Answering on Graphs,” Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD), pp. 99-110, 2010.
2. J. Sankara narayanan and H. Samet, “Query Processing Using Distance Oracles for Spatial Networks,” IEEE Trans. Knowledge and Data Eng., vol. 22, no. 8, pp. 1158-1175, Aug. 2010.
3. J. Sankaranarayanan, H. Samet, and H. Alborzi, “Path Oracles for Spatial Networks,” Proc. VLDB Endowment, vol. 2, no. 1, pp. 1210- 1221, 2009.
4. H. Samet, J. Sankaranarayanan, and H. Alborzi, “Scalable Network Distance Browsing in Spatial Databases,” Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD), pp. 43-54, 2008..
5. L. W.u, X. Xiao, D. Deng, G. Cong, A.D. Zhu, and S. Zhou, “Shortest Path and Distance Queries on Road Networks: An Experimental Evaluation,” Proc. VLDB Endowment, vol. 5, no. 5, pp. 406-417, 2012

BIOGRAPHY

Venkateswarlu CH is a Assistant Professor in the Department of Master of Computer Applications, Narayana Engineering College, Nellore. His research interest in Online Shortest Path computation based on live traffic index.

Sriram B.V, completed master of computer applications in narayana engineering college, Nellore. His research interest is Online Shortest Path computation based on live traffic index.

Sindhu.V, completed master of computer applications in narayana engineering college, Nellore. Her research interest is Online Shortest Path computation based on live traffic index.

.