International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

Live Activity List Based Shortest Path Calculation

[1]M.Arjun & [2]K.Sirisha
[1] M.Tech Student, Department of CSE, CREC, Tirupati,
Email id: hiarjun.m@gmail.com).
[2] Assistant Professor, Department of CSE, , CREC, Tirupati, INDIA,
Email.id: sirisha535@gmail.com

Abstract—
When you drive to somewhere 'far away', you will leave your current location via one of only a few 'important' traffic junctions. Starting from this informal observation, we develop an algorithmic approach—transit node routing— that allows us to reduce quickest-path queries in road networks to a small number of table lookups. We present two implementations of this idea, one based on a simple grid data structure and one based on highway hierarchies. For the road map of the United States, our best query times improve over the best previously published figures by two orders of magnitude. Our results exhibit various trade-offs between average query time (5 μs to 63 μs), preprocessing time (59min to 1200min), and storage overhead (21 bytes/node to 244 bytes/node).

***Index terms*:** Shortest path, air index, broadcasting

I INTRODUCTION

The classical way to compute the shortest path between two given nodes in a graph with given edge lengths is Dijkstra's algorithm Shortest path computation is an important function in modern car navigation systems and has been extensively studied in 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. 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. 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. 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 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 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.

1.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 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

2.LTI efficiently maintains the index for live traffic circumstances by incorporating Dynamic Shortest Path Tree (DSPT) [into hierarchical index techniques. In addition, a bounded version of DSPT is proposed to further reduce the broadcast overhead.

3.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 attempts to minimize all these performance factors for OSP.

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

II PRELIMINARY

A Performance Factors

The main performance factors involved in OSP are: (i) tune-in cost (at client side), (ii) broadcast size (at server side), and (iii) maintenance time (at server side), and (iv) query response time (at client side). In this work, we prioritize the tune-in cost as the main optimized factor since it affects the duration of client receivers into active mode and power consumption is essentially determined by the tuning cost If we minimize the tune-in cost of one service, then we reserve more resources for other services.

a shortest route using pre-stored weights

b .shortest route using traffic live (by LTI)

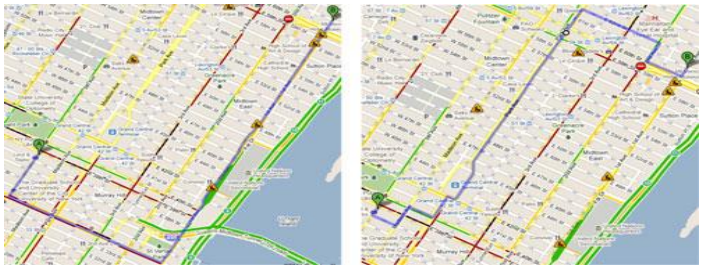


Fig. 1. Two alternative shortest paths

The index maintenance time and broadcast size relate to the freshness of the live traffic information. The maintenance time is the time required to update the index according to live traffic information. The broadcast size is relevant to the latency of receiving the latest index information. As the freshness is one of our main design criteria, we must provide reasonable costs for these two factors The last factor is the response time at client side. Given a proper index structure, the response time of shortest path computation can be very fast The computation also consumes power but their effect is outweighed by communication. It remains, however, an evaluated factor for OSP. BA Adaptation of Existing Approaches The communication cost is proportional to the number of clients. Thus, we omit this model from the remaining discussion

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. 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 reduces the search space by executing Dijkstra's algorithm simultaneously forwards from s and backwards from *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 and arc flags (AF) are two representative algorithms in this category

Dynamic shortest path tree **(DSPT)** maintains a tree structure Finding a shortest path from s to any node is computed at time on the shortest path tree. In their work, a simple dynamic version of Dijkstra is proposed which can outperform all competitors. locally for efficient shortest path retrieval

2 Index Transmission Model

The index transmission model enables servers to broadcast an index instead of raw traffic data. We review the state-of the-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.

Hierarchical index structures provide another way to abstracting and structuring a topographical index in a hierarchical fashion. Hierarchical MulTi-graph model (HiTi) is a representative approach in this category. Hierarchical encoded path view (HEPV) and Hub indexing share the same intuition of HiTi which divides large graph into smaller sub graphs and organize them in a hierarchical fashion by pushing up border nodes Furthermore, there is no discussion on how to maintain the TEDI structure in presence of edge weight updates.

Oracle focus on pre computing certain shortest path distances called oracles in order to answer approximate shortest path queries efficiently. Full pre-computation pre-computes the shortest paths between any two nodes in the road network, such as SILC and distance index . Even though these approaches offer fast query response time, the maintenance cost and size overhead become prohibitive on large road networks. Combination approaches integrate promising features from different index structure to support efficient shortest path computation. SHARC and CALT are two well studied combination approaches which integrate road map hierarchical approaches with AF and ALT, respectively

III LTI OVERVIEW AND OBJECTIVES

A LTI Overview

The traffic provider collects the live traffic circumstances from the traffic monitors via techniques like road sensors and traffic video analysis. The service provider periodically receives live traffic updates from the traffic provider and broadcasts the live traffic index on radio or wireless network.

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

When a mobile client wishes to compute and monitor a shortest path, it listens to the live traffic index and reads the relevant portion of the index for computing the shortest path.

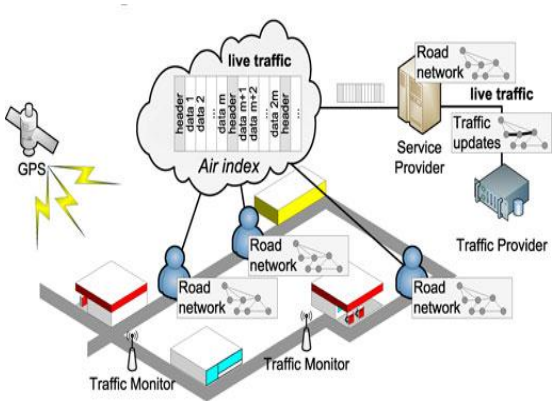


Fig. 3. LTI system overview

In Fig. 4, we illustrate the components and system flow in our LTI framework. The components shaded by gray color are the core of LTI. In order to provide live traffic information, the server maintains (component a) and broadcasts (component b) the index according to the up-to-date traffic circumstances. In order to compute the online shortest path, a client listens to the live traffic index, reads the relevant portions of the index (component c), and computes the shortest path (component d).

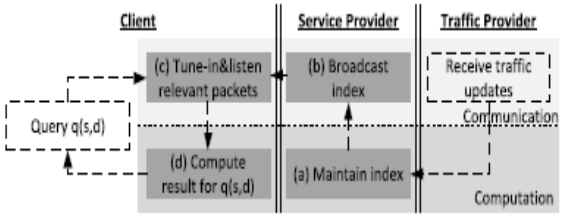


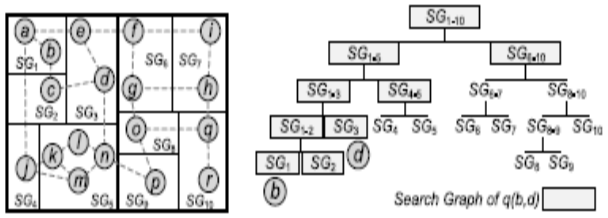
Fig 4. Components in LTI

B LTI Objectives

(1) Efficient maintenance strategy. Without efficient maintenance strategy, long maintenance time is needed at server side so that the traffic information is no longer live. This can reduce the maintenance time spent at component a.

(2) Light index overhead. The index size must be controlled in a reasonable ratio to the entire road map data. This reduces not only the length of a broadcast cycle, but also makes clients listen fewer packets in the broadcast channel. This can save the communication cost at components b and c.

(3) Efficient computation on a portion of entire index. This property enables clients to compute shortest path on a portion of the entire index. The computation at component d gets improved since it is executed on a smaller graph. This property also reduces the amount



(a) road network (b) hierarchical tree view

Fig. 5. Hierarchical index structure

Of data. hierarchical index structures enable clients to compute the shortest path on a portion of entire index. However, without pairing up with the first and second features, the communication and computation costs are still infeasible for OSP. the hierarchical index structures enable clients to compute the shortest path on a portion of entire index. However, without pairing up with the first and second features, the communication and computation costs are still infeasible for OSP.

IV LTI CONSTRUCTION

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.

A Analysis of Hierarchical Index Structures

Hierarchical index structures TEDI enable fast shortest path computation on a portion of entire index which significantly reduces the tune-in cost on the index transmission model. 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. Stochastic based index construction. The graph partitioning frame work only returns a binary tree index that is constructed based on Cheeger cut sequences.

International Journal of Research
International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

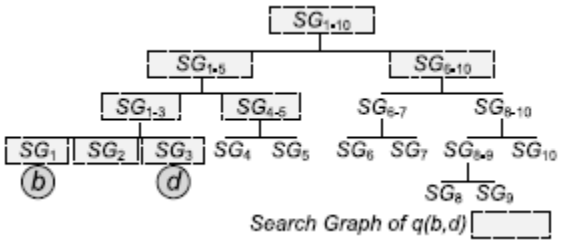


Fig.6.Effect of hierarchical structure.

To estimate the average size of the search graphs, we apply a stochastic process, Monte Carlo, that relies on random sampling to obtain numerical results.

The construction terminates when we have enough leave entries (i.e., g). Algorithm 1 shows the pseudo codes of the partitioning algorithm based on the stochastic process.

The effect of g. In this work, LTI requires only one parameter g to construct the index which is used to control the number of subgraphs being constructed. Our proposed techniques attempt to optimize the index (O2 and O3) subject to g.

V LTI TRANSMISSION

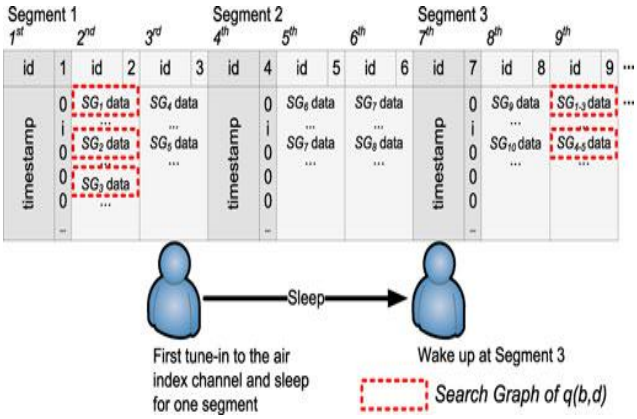
The broadcasting model uses radio or wireless network (e.g., 3G, LTE, Mobile WiMAX) as the transmission medium. When the server broadcasts a data set (i.e., a "programme"), all clients can listen to the data set concurrently. Thus, this transmission model scales well independent of the number of clients. A broadcasting scheme is a protocol to be followed by the server and the clients. First, the server partitions the data set into m equi-sized data segments. Each packet contains a header and a data segment, where a header describes the broadcasting schedule of all packets. In this example, the variables i and n in each header represent the last broadcasted item and the total number of items. The server periodically broadcasts a sequence of packets. The query performance can be measured by the tuning time and the waiting time at the client side. The tuning time is the time for reading the packets. The waiting time is the time from the start time to the termination time of the query. In this broadcasting scheme, the parameter m decides the tradeoff between tune-in size and the overhead. A large m favors small tune-in size whereas a small m incurs small waiting time. Imielinski et al. suggests to set m to the square root of the ratio of the data size to the index size.

TABLE 2
Packet Format on the Air Index

Offset	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	id				checksum				...							
...	...															

5.3 Client Tune-in Procedures of Air LTI

Fig. 9 shows the content of a broadcast cycle for a LTI structure in Fig. 7. In this example, the air index uses a ð1; 2Þ interleaving scheme and each data packet stores the edge



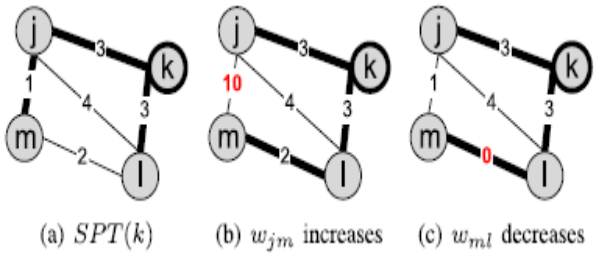
weight of different sub graphs. For instance, the edge weight of sub graph SG1 are stored in the 2nd packet of a broadcast cycle.

Fig. 7: Receiving LTI data from the air index

VI .LTI MAINTENANCE

In this section, we study an incremental update approach that can efficiently maintain the live traffic index according to the updates. As a remark, the entire update process is done at the service provider and there is no extra data structure being broadcasted to the clients. To reduce the maintenance cost, we incorporates dynamic shortest path tree technique (DSPT) into the hierarchical index structures and reduce the size of trees by a bounded version (BSPT) The weight of these shortcuts can be maintained by the corresponding shortest path tree from each border node BSGi . Given a graph G ¼ ðV ;EÞ, a shortest path tree (SPT) rooted at a vertex r 2 V, denoted as SPTðrÞ, is a tree with root r, and 8v 2 V _ frg, SPTðrÞ contains a shortest path from r to v. In Fig. 10a, the shortest path tree of vertex k is highlighted by bold lines. Given a shortest path tree, a dynamic Dijkstra approach [22] is proposed for handling both weight increasing (Fig. 10b) and decreasing cases (Fig. 10c). The intuition of the algorithms is to find the affected local vertices and revise the shortest path tree using a Dijkstra like algorithm starting from the updated vertices. For instance, the weight of edgeðm; lÞ is decreased from 2 to 0. Starting from the vertex m, a new path m ! l ! k, that is a better path from m to k, is found by the Dijkstra searching. Thereby, the update process revises the shortest path tree accordingly as shown in Fig. 10c.

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org



(a) $SPT(k)$ (b) w_{jm} increases (c) w_{ml} decreases

Fig. 8:. Shortest path tree maintenance

Definition 1 (Bounded shortest path tree (BSPT)).

Inspired by the discussion, we propose a variant shortest path tree, named as bounded shortest path tree BSPTðvÞ, in Definition 1. A shortest path starting from v is necessarily kept in a bounded shortest path tree BSPTðvÞ if and only if the distance of the shortest path is shorter than the distance from v to every border node. In Fig. 11b, BSPTðlÞ keeps only one shortest path l ! k. The shortest path l ! m and l ! j are dropped since the shortest distance of l ! m and l ! j is not shorter than the distance from l to border node k. Typically, BSPTðvÞ is much smaller than SPTðvÞ and it also boosts up computation efficiency due to smaller search space. Pruning ability of BSPT. The pruning ability of BSPT is highly relevant to the border node selection in each subgraph. In the worst case, BSPT performs as the same as a na€ıve SPT if the borders are very far from each others. However, such cases rarely happen in LTI since the graph partitioning technique prefers a partitioning having small number of borders, which minimizes the change of the worst-case scenario. In our study, BSPT prunes 30 to 50 percent edges from the complete SPT for our evaluated data sets.

Algorithm 1 Stochastic Partitioning Algorithm

PQ: a priority queue; I: index structure;
Algorithm $partition(G$:the graph, γ:the number of partitions)
1: $(\lambda, \mathcal{V}) := eigen(G)$ and $n :=$ root of I
2: insert $(n, G, \mathcal{V}, \lambda)$ into PQ in decreasing order to λ
3: **while** $|PQ| < \gamma$ **do**
4: $(n, G, \mathcal{V}, \lambda) := PQ.pop()$
5: **for** k:=2 to $\gamma - |PQ| + 1$ **do**
6: decompose G into $SG_1 ... SG_k$ s.t. Eq. 4 is minimized
7: form a temporal index I' that attaches $SG_1 ... SG_k$
8: **if** avg($\mathcal{S}(I')$) is better than $best_{\mathcal{S}}$ **then**
9: update $best_{\mathcal{S}}$ and $best_{SG} := \{SG_1, ..., SG_k\}$
10: attach $best_{SG}$ as n's children
11: **for** i:=1 to $|best_{SG}|$ **do**
12: insert $(n_i, SG_i, \mathcal{V}_i, \lambda_i)$ into PQ
13: **return** I

Algorithm 2 Client Algorithm

Algorithm $Client(I$:LTI, s:source, t:destination)
1: generate G^q from I based on s and d
2: listen to the channel for a header segment
3: read the header segment ▷ Section 5.3
4: decide the necessary segments to be read ▷ Section 5.3
5: wait for those segments, read them to update the weight of G^q
6: compute the shortest path (from s to t) on G^q ▷ Section 4.1

Algorithm 3 Service Algorithm

Algorithm $Service(G$:graph)
1: construct I and $\{SG_i\}$ based on G ▷ Section 4.2
2: **for** each broadcast cycle **do**
3: collect traffic updates from the traffic provider
4: update the subgraphs $\{SG_i\}$ ▷ Section 6
5: broadcast the subgraphs $\{SG_i\}$ ▷ Section 5.2

PUTTING ALL TOGETHER

A client can invoke Algorithm 2 in order to find the shortest path from a source s to a destination t. First, the client generates a search graph Gq based on s (i.e., current location) and d. When the client tunes-in the broadcast channel it keeps listening until it discovers a header segment (cf. Fig. 9). After reading the header segment, it decides the necessary segments (to be read) for computing the shortest path. These issues are addressed. The client then waits for those segments, reads them, and updates the weight of Gq. Subsequently, Gq is used to compute the shortest path in the client machine locally Note that Algorithm 2 is kept running in order to provide online shortest path until the client reaches to the destination. service provider, as shown in Algorithm 3. The first step is devoted to construct the live traffic index; they are offline tasks to be executed once only. The service provider builds the live traffic index by partitioning the graph G into a set of sub graphs fSGig such that they are ready for broadcasting. We develop an effective graph partitioning algorithm for minimizing the total size of sub graphs and study a combinatorial optimization for reducing the search space of shortest path queries in. In each broadcasting cycle, the server first collects live traffic updates from the traffic provider, updates the sub graphs fSGig eventually broadcasts them.

VII 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.

REFERENCE

[1] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes, "In Transit to Constant Time Shortest-Path Queries in Road Networks," Proc. Workshop Algorithm Eng. and Experiments (ALENEX), 2007.

[2] P. Sanders and D. Schultes, "Engineering Highway Hierarchies," Proc. 14th Conf. Ann. European Symp. (ESA), pp. 804-816, 2006.

[3] G. Dantzig, Linear Programming and Extensions, series Rand Corporation Research Study Princeton Univ. Press, 1963.

[4] R.J. Gutman, "Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks," Proc. Sixth Workshop Algorithm Eng. and Experiments and the First Workshop Analytic Algorithmics and Combinatorics (ALENEX/ANALC), pp. 100-111, 2004.

[5] B. Jiang, "I/O-Efficiency of Shortest Path Algorithms: An Analysis," Proc. Eight Int'l Conf. Data Eng. (ICDE), pp. 12-19, 1992.

[6] P. Sanders and D. Schultes, "Highway Hierarchies Hasten Exact Shortest Path Queries," Proc. 13th Ann. European Conf. Algorithms (ESA), pp. 568-579, 2005.

[7] D. Schultes and P. Sanders, "Dynamic Highway-Node Routing," Proc. Sixth Int'l Conf. Experimental Algorithms (WEA), pp. 66-79, 2007.

[8] F. Zhan and C. Noon, "Shortest Path Algorithms: An Evaluation Using Real Road Networks," Transportation Science, vol. 32, no. 1, pp. 65-73, 1998.

[9] "Google Maps," http://maps.google.com, 2014.

[10] "NAVTEQ Maps and Traffic," http://www.navteq.com, 2014.

[11] "INRIX Inc. Traffic Information Provider," http://www.inrix. com, 2014.

[12] "TomTom NV," http://www.tomtom.com, 2014.

[13] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010-2015," 2011.

[14] D. Stewart, "Economics of Wireless Means Data Prices Bound to Rise," The Global and Mail, 2011.

[15] W.-S. Ku, R. Zimmermann, and H. Wang, "Location-Based Spatial Query Processing in Wireless Broadcast Environments," IEEE Trans. Mobile Computing, vol. 7, no. 6, pp. 778-791, June 2008.