International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Online Shortest Path Computation

*K.Saresh **A.Madhavi

*M.TECH student, Dept of CSE, VAAGDEVI COLLEGE OF ENGINEERING
**Assistant Professor, Dept of CSE, VAAGDEVI COLLEGE OF ENGINEERING

Abstract:

Online Shortest path computation using live traffic index in road networks aims at computing the shortest path from source to destination using Live traffic index. The problem of finding the shortest path between two intersections on a road map (the graph's vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of its road segment) may be modeled by a special case of the shortest path problem in graphs. In this paper, index transmission model along with Live traffic index technique is used. In this technique, first the traffic provider collects the traffic data and transmits to the traffic server broadcaster. The traffic server broadcaster indexes and optimizes the traffic data and broadcasts to the navigation client. Graph partitioning and stochastic process are the new techniques used to optimize the index. The fast query response time and short tune in cost at client side, small broadcast size and maintainence time at server side are the extra features achieved in the system. 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

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 TomTom 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 con-tinuous 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. Suppose you would like to find the shortest path between two intersections on a city map: a starting point and a destination. Dijkstra's algorithm initially marks the distance (from the starting point) to every other intersection on the map with infinity. This is done not to imply there is an infinite distance, but to note that those intersections have not yet been visited; some variants of this method simply leave the intersections' distances unlabeled. Now, at each iteration, select the current intersection. For the first iteration, the current intersection will be the starting point, and the distance to it (the intersection's label) will be zero. For subsequent iterations (after the first), the current intersection will be the closest unvisited intersection to the starting point (this will be easy to find).

From the current intersection, update the distance to every unvisited intersection that is directly connected to it. This is done by determining the sum of the distance between an unvisited intersection and the value of the current intersection, and relabeling the unvisited intersection with this value (the sum), if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each neighboring intersection, mark the current intersection as visited, and select the unvisited intersection with lowest distance (from the starting point) – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned to. Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited(as is the case with any visited intersection) you have determined the shortest path to it, from the starting point, and can trace your way

back, following the arrows in reverse; in the algorithm's implementations, this is usually done (after the algorithm has reached the destination node) by following the nodes' parents from the destination node up to the starting node; that's why we keep also track of each node's parent. This algorithm makes no attempt to direct "exploration" towards the destination as one might expect. Rather, the sole consideration in determining the next "current" intersection is its distance from the starting point. This algorithm therefore expands outward from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the algorithm necessarily finds the shortest path. However, it may also reveal one of the algorithm's weaknesses: its relative slowness in some topologies. If one represents a nondeterministic abstract machine as a graph where vertices describe states and edges describe possible transitions, shortest path algorithms can be used to find an optimal sequence of choices to reach a certain goal state, or to establish lower bounds on the time needed to reach a given state. For example, if vertices represent the states of a puzzle like a Rubik's Cube and each directed edge corresponds to a single move or turn, shortest path algorithms can be used to find a solution that uses the minimum possible number of moves.

In a networking or telecommunications mindset, this shortest path problem is sometimes called the min-delay path problem and usually tied with a widest path problem. For example, the algorithm may seek the shortest (min-delay) widest path, or widest shortest (min-delay) path. A more light-hearted application is the games of "six degrees of separation" that try to find the shortest path in graphs like movie stars appearing in the same film.

In a networking or telecommunications mindset, this Other applications, often studied in operations research, include plant and facility layout, robotics, transportation, and VLSI design. 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 broad-casted 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

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

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 net-work 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 pairwise mini-mum 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 as shown in Fig.1


(a) Shortest route using pre-stored weights (b) Shortest route using live traffic (by LTI)

Fig. 1. Two alternative shortest paths in Manhattan, NY.

II. SYSTEM PREPARATIONS

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 (i.e., number of packets received) [17], [23]. In addition, shortening the duration of active mode enables the clients to receive more services simultaneously by selective tuning [24]. These services may include providing live weather information, delivering latest promotions in surrounding area, and monitoring avail-ability of parking slots at

destination. If we minimize the tune-in cost of one service, then we reserve more resources for other services. 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 (i.e., few milliseconds on large road maps) which is negligible compared to access latency for current wireless network speed. The computation also consumes power but their effect is outweighed by communication. It remains, however, an evaluated factor for OSP.

Adaptation of Existing Approaches: In this section, we briefly discuss the applicability of the state-of-the-art shortest path solutions on different trans-mission 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.

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) [16] are two representative algorithms in this category. ALT makes use of A_ search, landmarks, and triangle inequality [17]. 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 [18] 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 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 sub-graph 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 [19]. 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 [12] discusses how to maintain a correct shortest path tree rooted at s after receive a set of edge weight updates to the graph. 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 pro-posed which can outperform all competitors.

III. 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

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

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) [10], and transit-node 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. The length of the shortest path ðs; tÞ that passes at least one transit node is given all involved distances can be directly looked up in the pre-computed data structure. Note that if the shortest path that passes no transit node, then other shortest path algorithm is applied instead. The hierarchical approaches can provide very fast query time as reported in [11]. 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 [12], 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.

IV. LTI OVERVIEW AND OBJECTIVES

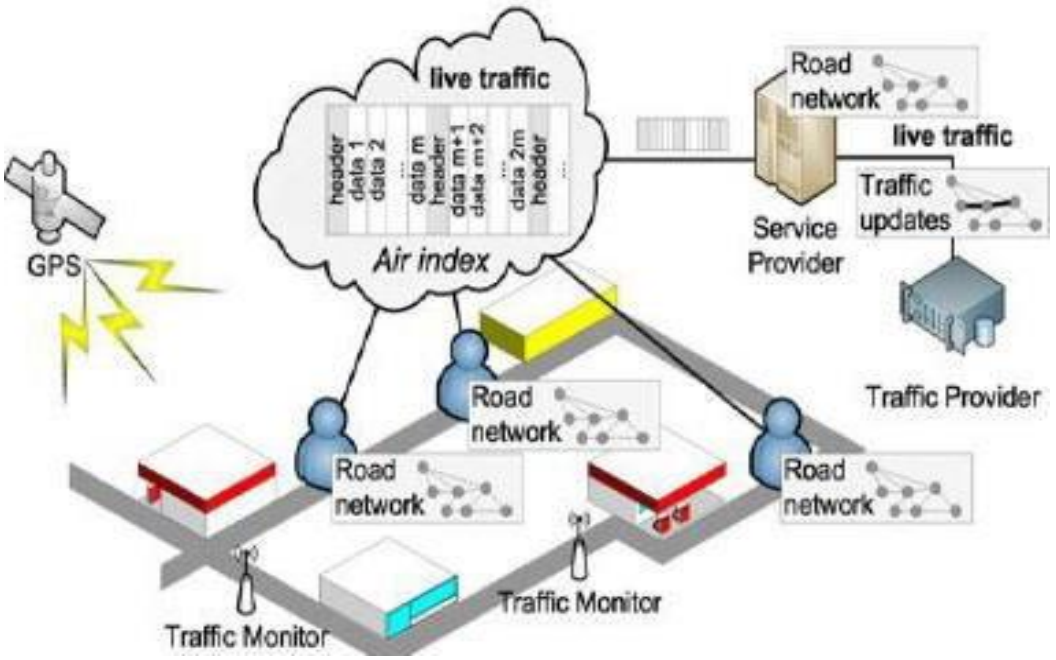
A. LTI Overview

A road network monitoring system typically consists of a service provider, a large number of mobile clients (e.g., vehicles), and a traffic provider (e.g., GoogleMap, NAV-TEQ, INRIX, etc.). Fig. 2 shows an architectural overview of

this system in the context of our live traffic index frame-work. The traffic provider collects the live traffic circumstances from the traffic monitors via techniques like road

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

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 (e.g., 3G, LTE, Mobile WiMAX, etc.). 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 live traffic circumstances). Thus, we assume that the graph structures are distributed to every client in advance (e.g., by monthly updates or at system boot-up) via typical transmission protocol (i.e., HTTP and FTP).



In Fig. 2, 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).

LTI Objectives: To optimize the performance of the LTI components, our solution should support the following features.

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.

Light Index Overhead: The index size must be con-trolled 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.

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 of data received and energy consumed at component c. Inspired by these properties, LTI has relatively short tune-in cost (at client side), fast query response time (at client side), small broadcast size (at server side), and light.

V. LTI TRANSMISSION

A. Broadcasting Scheme

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 concur-rently. 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. The (1,m) interleaving scheme [20] is one of the best broadcasting schemes. Table 1 shows an example broad-casting cycle with m ¼ 3 packets and the entire data set contains six data items. First,

the server partitions the data set into m equisized 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 broad-casted item and the total number of items. The server periodically broadcasts a sequence of packets (called as a broadcast cycle). We use a concrete example to demonstrate how a client receives her data from the broadcast channel. Suppose that a client wishes to query for the data object o5. First, the client tunes in the broadcast channel and waits until the next header is broadcasted. For instance, the client is listening to the header of the first packet, and finds out that the third packet contains o5. In order to preserve energy, the client sleeps until the broadcasting time of that packet. Then, it wake-ups and reads the requested data item from the packet. 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 over-head. A large m favors small tune-in size whereas a small m incurs small waiting time. Imielinski et al. [20] suggests to set m to the square root of the ratio of the data size to the index size.

VI. PUTTING ALL TOGETHER

We are now ready to present our complete LTI frame-work, which integrates all techniques been discussed. 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 (cf. Section 5.2), 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 in Section 5.3. The client then waits for those segments, reads them, and update 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. We then discuss about the tasks to be performed by the 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 subgraphs fSGi g such that they are ready for broadcasting. We develop an effective graph partitioning algorithm for minimizing the total size of subgraphs and study a combinatorial optimization for reducing the search space of shortest path queries in Section 4.2. In each broadcasting cycle, the server first collects live traffic updates from the traffic provider, updates the subgraphs fSGig (discussed in Section 6), and 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.

VIII. REFERENCES

[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 Cor-poration 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 Analy-sis,‖ 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, 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 Fore-cast 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.

[16]N. Malviya, S. Madden, and A. Bhattacharya, ―A Continuous Query System for Dynamic Route Planning,‖ Proc. IEEE 27th Int'l Conf Data Eng. (ICDE), pp. 792-803, 2011.

[17]G. Kellaris and K. Mouratidis, ―Shortest Path Computation on Air Indexes,‖ Proc. VLDB Endowment, vol. 3, no. 1, pp. 741-757, 2010.

[18]Y. Jing, C. Chen, W. Sun, B. Zheng, L. Liu, and C. Tu, ―Energy-Efficient Shortest Path Query Processing on Air,‖ Proc. 19th ACM SIGSPATIAL Int'l Conf. Advances in Geographic Information Systems (GIS), pp. 393-396, 2011.

[19]R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina, ―Proximity Search in Databases,‖ Proc. Int'l Conf. Very Large Databases (VLDB), pp. 26-37, 1998.

[20]N. Jing, Y.-W. Huang, and E.A. Rundensteiner, ―Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation,‖ IEEE Trans. Knowledge and Data Eng., vol. 10, no. 3, pp. 409-432, May 1998.

[21]S. Jung and S. Pramanik, ―An Efficient Path Computation Model for Hierarchically Structured Topographical Road Maps,‖ IEEE Trans. Knowledge and Data Eng., vol. 14, no. 5, pp. 1029-1046, Sept. 2002.

AUTHOR 1 :-

*K.Saresh completed his B tech in vaagdevi engineering college in 201 and pursuing M-Tech in Vaagdevi College of Engineering

AUTHOR 2:-

**A.Madhavi is working as Assistant Professor in Dept of CSE, Vaagdevi College of Engineering.