ISSN:0975-9646

R.Kamatchi et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (2) , 2015, 1526-1534

An Automated Framework for Shortest Path Computation

R.Kamatchi,
Research Scholar,
Mother Teresa Women's University,
Kodaikanal, India

Dr.S.Lakshmi,
Associate Professor,
Department of Computer Science and Engineering,
Jeppiaar Engineering College, Chennai India

Abstract:Shortest path computation is one of the most common queries in location-based services that involve transportation networks. Motivated by scalability challenges faced in the mobile network industry, we propose adopting the wireless broadcast model for such location-dependent applications. Existing work develop a 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. As such, huge communication cost will be spent on sending result paths on this model. 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.

I.INTRODUCTION:

Computing fastest routes in road networks is one of the showpieces of real-world applications of algorithmic. In principle we could use Dijkstra's algorithm. But for large road networks this would be far too slow. Therefore, in recent years, there has been considerable interest in speed-up techniques for route planning. This makes it possible to preprocess some information once and for all that can be used to accelerate all subsequent point-to-point queries. Today, the shortest path computation problem in road networks can be regarded as largely solved. However, real road networks change all the time. In this paper, we address two such dynamic scenarios: We present two novel methods, namely Elliptic Boundary (EB) and Next Region (NR)e.g., due to traffic jams, and switching between different cost functions that take vehicle type, road restrictions, or driver preferences into account.

For example, in Google Maps, once a shortest path from A to B has been obtained which passes through C, users can simply change the query to find the shortest path from A to B which is constrained to pass through D instead of C and the new shortest path is presented to the user instantly. Requiring that the result be obtained in real time (or almost real time) precludes the use of conventional algorithms that are graph-based (e.g., the INE and IER methods and improvements) which usually incorporate Dijkstra's algorithm in at least some parts of the solution

LITERATURE SURVEY:

IN TRANSIT TO CONSTANT TIME SHORTEST-PATH QUERIES IN ROAD NETWORKS

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 (59 min to 1200 min), and storage overhead (21 bytes/node to 244 bytes/node).

ENGINEERING HIGHWAY HIERARCHIES

We presented a shortest path algorithm that allows fast point-to-point queries in graphs using preprocessed data. Here, we give an extensive revision of our method. It allows faster query and preprocessing times, it reduces the size of the data obtained during the preprocessing and it deals with directed graphs. Some important concepts like the neighbourhood radii and the contraction of a network have been generalized and are now more flexible. The query algorithm has been simplified: it differs only by a few lines from the bidirectional version of DIJKSTRA'S algorithm. We can prove that our algorithm is correct even if the graph contains several paths of the same length.

Experiments with real-world road networks confirm the effectiveness of our approach. Preprocessing the network of Western Europe, which consists of about 18 million nodes, takes 15 minutes and yields 68 bytes of additional data per node. Then, random queries take 0.76 ms on average. If we are willing to accept slower query times (1.38 ms), the memory usage can be decreased to 17 bytes per node. For the European and the US road networks, we can guarantee that at most 0.05% of all nodes are visited during any query.

REACH-BASED ROUTING: A NEW APPROACH TO SHORTEST PATH ALGORITHMS OPTIMIZED FOR ROAD NETWORKS

We study the point-to-point shortest path problem in a setting where preprocessing is allowed. We improve the reach-based approach of Gutman in several ways. In particular, we introduce a bidirectional version of the algorithm that uses implicit lower bounds and we add shortcut arcs which reduce vertex reaches. Our modifications greatly reduce both preprocessing and query times. The resulting algorithm is as fast as the best previous method, due to Sanders and Schultes. However, our

algorithm is simpler and combines in a natural way with A∗ search, which yields significantly better query times.

I/O-EFFICIENCY OF SHORTEST PATH ALGORITHMS: AN ANALYSIS

To establish the behavior of algorithms in a paging environment, the author analyzes the input/output (I/O) efficiency of several representative shortest path algorithms. These algorithms include single-course, multisource, and all pairs ones. The results are also applicable for other path problems such as longest paths, most reliable paths, and bill of materials. The author introduces the notation and a model of a paging environment. The I/O efficiencies of the selected single-source, all pairs, and multisource algorithms are analyzed and discussed.

HIGHWAY HIERARCHIES HASTEN EXACT SHORTEST PATH QUERIES

We present a new speedup technique for route planning that exploits the hierarchy inherent in real world road networks. Our algorithm preprocesses the eight digit number of nodes needed for maps of the USA or Western Europe in a few hours using linear space. Shortest (i.e. fastest) path queries then take around eight milliseconds to produce exact shortest paths. This is about 2 000 times faster than using Dijkstra's algorithm.

DYNAMIC HIGHWAY-NODE ROUTING

We introduce a dynamic technique for fast route planning in large road networks. For the first time, it is possible to handle the practically relevant scenarios that arise in present-day navigation systems: When an edge weight changes (e.g., due to a traffic jam), we can update the preprocessed information in 2-40ms allowing subsequent fast queries in about one millisecond on average. When we want to perform only a single query, we can skip the comparatively expensive update step and directly perform a prudent query that automatically takes the changed situation into account. If the overall cost function changes (e.g., due to a different vehicle type), re computing the preprocessed information takes typically less than two minutes.

The foundation of our dynamic method is a new static approach that generalises and combines several previous speedup techniques. It has outstandingly low memory requirements of only a few bytes per node.

SHORTEST PATH ALGORITHMS: AN EVALUATION USING REAL ROAD NETWORKS

The classic problem of finding the shortest path over a network has been the target of many research efforts over the years. These research efforts have resulted in a number of different algorithms and a considerable amount of empirical findings with respect to performance. Unfortunately, prior research does not provide a clear direction for choosing an algorithm when one faces the problem of computing shortest paths on real road networks. Most of the computational testing on shortest path algorithms has been based on randomly generated networks, which may not have the characteristics of real road networks. In this paper, we provide an objective evaluation of 15 shortest path algorithms using a variety of real road networks. Based on the evaluation, a set of recommended algorithms for computing shortest paths on real road networks is identified. This evaluation should be particularly useful to researchers and practitioners in operations research, management science, transportation, and Geographic Information Systems. The computation of shortest paths is an important task in many network and transportation related analyses. The development, computational testing, and efficient implementation of shortest path algorithms have remained important research topics within related disciplines such as operations.

LOCATION-BASED SPATIAL QUERY PROCESSING IN WIRELESS BROADCAST ENVIRONMENTS

Location-based spatial queries (LBSQs) refer to spatial queries whose answers rely on the location of the inquirer. Efficient processing of LBSQs is of critical importance with the ever-increasing deployment and use of mobile technologies. We show that LBSQs have certain unique characteristics that traditional spatial query processing in centralized databases does not address. For example, a significant challenge is presented by wireless broadcasting environments, which have excellent scalability but often exhibit high-latency database access. In this paper, we present a novel query processing technique that, while maintaining high scalability and accuracy, manages to reduce the latency considerably in answering location-based spatial queries. Our approach is based on peer-to-peer sharing, which enables us to process queries without delay at a mobile host by using query results cached in its neighboring mobile peers. We demonstrate the feasibility of our approach through a probabilistic analysis, and we illustrate the appeal of our technique through extensive simulation results.

SHORTEST PATH COMPUTATION ON AIR INDEXES

Shortest path computation is one of the most common queries in location-based services that involve transportation networks. Motivated by scalability challenges faced in the mobile network industry, we propose adopting the wireless broadcast model for such location-dependent applications. In this model the data are continuously transmitted on the air, while clients listen to the broadcast and process their queries locally. Although spatial problems have been considered in this environment, there exists no study on shortest path queries in road networks. We develop the first framework to compute shortest paths on the air, and demonstrate the practicality and efficiency of our techniques through experiments with real road networks and actual device specifications.

ENERGY- EFFICIENT SHORTEST PATH QUERY PROCESSING ON AIR

Wireless broadcast provides a scalable and secure spatial data dissemination approach for geographical applications in wireless mobile environments. Among various location-based services, the shortest path query on road networks is

one of the most popular and essential services in our daily life. In this paper, we propose an energy-efficient scheme for on air shortest path query processing on road networks, which leverages an elaborate air index called Bag Index based upon the novel Hilbert-based heuristic tree decomposition for the road networks. Experimental results show that the proposed approach incurs less energy consumption on both communication and computation than the previous schemes.

PROXIMITY SEARCH IN DATABASES

An information retrieval (IR) engine can rank documents based on textual proximity of keywords within each document. In this paper we apply this notion to search across an entire database for objects that are \near " other relevant objects. Proximity search enables simple \focusing " queries based on general relationships among objects, helpful for interactive query sessions. We view the database as a graph, with data in vertices (objects) and relationships indicated by edges. Proximity is dened based on shortest paths between objects. We have implemented a prototype search engine that uses this model to enable keyword searches over databases, and we have found it very e active for quickly ending relevant information. Computing the distance between objects in a graph stored on disk can be very expensive. Hence, we show how to build compact indexes that allow us to quickly nd the distance between objects at search time. Experiments show that our algorithms are e-cient and scale well.

HIERARCHICAL ENCODED PATH VIEWS FOR PATH QUERY PROCESSING

Efficient path computation is essential for applications such as intelligent transportation systems (ITS) and network routing. In ITS navigation systems, many path requests can be submitted over the same, typically huge, transportation network within a small time window. While path pre computation (path view) would provide an efficient path query response, it raises three problems which must be addressed: 1) pre computed paths exceed the current computer main memory capacity for large networks; 2) disk-based solutions are too inefficient to meet the stringent requirements of these target applications; and 3) path views become too costly to update for large graphs (resulting in out-of-date query results). We propose a hierarchical encoded path view (HEPV) model that addresses all three problems. By hierarchically encoding partial paths, HEPV reduces the view encoding time, updating time and storage requirements beyond previously known path pre computation techniques, while significantly minimizing path retrieval time. We prove that paths retrieved over HEPV are optimal. We present complete solutions for all phases of the HEPV approach, including graph partitioning, hierarchy generation, path view encoding and updating, and path retrieval. In this paper, we also present an in-depth experimental evaluation of HEPV based on both synthetic and real GIS networks. Our results confirm that HEPV offers advantages over alternative path finding approaches in terms of performance and space efficiency

AN EFFICIENT PATH COMPUTATION MODEL FOR HIERARCHICALLY STRUCTURED TOPOGRAPHICAL ROAD MAPS

In this paper, we have developed a HiTi (Hierarchical MulTi) graph model for structuring large topographical road maps to speed up the minimum cost route computation. The HiTi graph model provides a novel approach to abstracting and structuring a topographical road map in a hierarchical fashion. We propose a new shortest path algorithm named SPAH, which utilizes HiTi graph model of a topographical road map for its computation. We give the proof for the optimality of SPAH. Our performance analysis of SPAH on grid graphs showed that it significantly reduces the search space over existing methods. We also present an in-depth experimental analysis of HiTi graph method by comparing it with other similar works on grid graphs. Within the HiTi graph framework, we also propose a parallel shortest path algorithm named ISPAH. Experimental results show that inter query shortest path problem provides more opportunity for scalable parallelism than the intra query shortest path problem.

II. RELATED WORK:

A) ROAD NETWORKS AND SHORTEST PATH QUERY

Algorithms for shortest path queries are categorized as follows: (a) those with no pre-computation, where only the road network information is available, and (b) those with pre-computation, where shortest paths between some or all node pairs are pre-calculated and appropriate information is materialized in order to speed up the search.

1) WITHOUT PRE-COMPUTATION:

A common algorithm is Dijkstra's [13]. Initially, nodes adjacent to vs are pushed into a min-heap with their graph weights from vs as sorting keys. The top node v in the heap is popped in every iteration and expanded, i.e., its adjacent nodes v 0 are en-heaped with key equal to that of v plus the weight of edge (v, v 0). The process stops when vt is popped. The shortest path is returned by tracing backwards the expansions that lead to vt. A* search [14] improves on Dijkstra's algorithm but requires a lower bound LB(v, vt) to be known for the graph distance between an encountered node v and the target node vt. The difference from Dijkstra is that the key of each en heaped node v is increased by LB(v, vt). We ignore A* in the following since we assume general road networks (where no a priori lower bounds exist).

2) WITH PRE-COMPUTATION:

First partitions the network nodes. A bit vector (flag)[15] is assigned to every edge, where each bit corresponds to a partition; in the flag of edge (vi, vj) the bit for a partition is 1 if there is at least one node v in the partition where the shortest path from vi to v traverses (vi, vj). Search (e.g., Dijkstra) only considers edges whose bit for vt's partition is 1. Landmark chooses some anchor nodes (called landmarks) and pre-computes for each node v its graph distances to all anchor nodes. A distance vector is then created from the distances to the anchor nodes. From the distance vectors of two nodes, a lower bound can be derived for their graph distance.

3) WIRELESS BROADCASTING AND AIR INDEXES

In this model the server repeatedly transmits identical broadcast cycles, each containing the entire database and potentially some indexing information (called air index). The broadcast cycle consists of fixed-size packets, defining the smallest information unit transmitted. The most common organization of the broadcast cycle is the (1,m) interleaving scheme

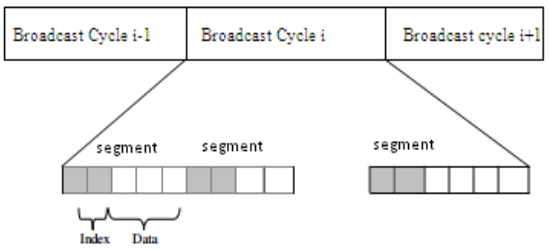

figure 1 **(1,*m*) scheme**

exemplified in Figure 1; the data tuples are placed into m equi-sized data segments interleaved by m copies of the index (e.g., a B-tree). To process a query, the client tunes in the broadcast channel and waits until the next index is broadcast; the larger m is, the shorter the wait for the index. The client receives the index, performs its point/range selection, and then waits until the data segments that contain the result tuples are broadcast; the larger m is, the longer the wait for the data.

III. PROBLEM STATEMENT

The shortest path computation is mainly used in the road network. The very first solution of the optimal route query is based on the pre-stored weights. The first approaches considered only the end points. Most of the techniques have limitations in some particular area. The main problems are to find shortest path during road traffic networks. Among this some may consider constraints and others without consider the constraints and these are the main problems to consider.

IV. MOTIVATION:

We are planning a weekend trip around the town as follows: first we intend to visit a shopping center in the afternoon to check the season's new arrivals, then we plan to take lunch in an Indian restaurant, and finally, we would like to watch a specific movie at late night. Naturally, we intend to drive the minimum overall distance to these destinations. That is, we need to find the locations of the shopping center si, the Indian restaurant rj, and the theater tk that shows our movie, where traveling between these locations in the given order would result in the shortest travel distance (or time).

We call this type of queries where the order of points to be visited is given and fixed, the optimal sequenced route queries or OSR for short. Using Fig. 2, we show that the OSR query cannot be optimally answered by simply performing a series of independent nearest neighbor searches from different locations. We use the first example described above as our running example throughout the paper. The figure shows a network of equally sized connected square cells, three different types of point sets shown by white, black and gray circles representing shopping centers, Indian restaurants, and theaters, respectively, and a starting point p (shown by fig1).

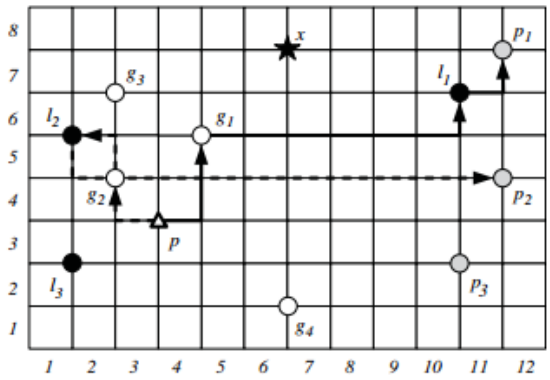

FIGURE 2

V. PRELIMINARY

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) . In addition, shortening the duration of active mode enables the clients to receive more services simultaneously by selective tuning . These services may include providing live weather information, delivering latest promotions in surrounding area, and monitoring availability 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.

Bidirectional Search executes Dijkstra's algorithm simultaneously forwards from the source s and backwards from the target t. Once some node has been visited from both directions, the shortest path can be derived from the information already gathered [4]. Many more advanced speed-up techniques use bidirectional search as an optional or sometimes even mandatory ingredient.

Hierarchical Approaches try to exploit the hierarchical structure of the given network. In a preprocessing step, a hierarchy is extracted, which can be used to accelerate all subsequent queries.

Many speed-up techniques can be combined. In [17], a combination of a special kind of geometric container [18], the separator-based multi-level method [19], and A∗ search yields a speed-up of 62 for a railway transportation problem. In [20], combinations of A∗ search, bidirectional search, the separator-based multi-level method, and geometric containers are studied: Depending on the graph type, different combinations turn out to be best

B)ADAPTATION OF EXISTING APPROACHES

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.

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) 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 and arc flags (AF) are two representative algorithms in this category.

ALT makes use of A search, landmarks, and triangle inequality. 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 search on the graph. Delling and Wagner propose 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 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 consumes too much space for large road networks. The dynamic updates of AF (DAF) have been recently studied in. 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. 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 time on the shortest path tree. In their work, a simple dynamic version of Dijkstra is proposed which can outperform all competitors.

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. These speed-up approaches include reach, highway hierarchies (HH) , contraction hierarchies (CH) , and transit node routing (TNR).

Reach, HH, and CH are based on shortcut techniques, 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.

The hierarchical approaches can provide very fast query time as reported in. 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 but the solution is limited to weight increasing cases. In, 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) 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) 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. However, both are infeasible for OSP since these approaches suffer from the excessive storage overhead for a large amount of pre-computed path information.

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. Besides, as reported by, the performance of the full pre-computation approaches (i.e., SILC is not much superior to those road map hierarchical approaches (i.e., CH.)

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. However, these complex index structures are either lack of efficient maintenance strategies or have huge size overhead which makes them inapplicable to OSP on any transmission model.

DISCUSSION

Except for hierarchical index structures, all methods on either raw or index transmission models suffer from a drawback that a few updates could affect a large portion of packets. We demonstrate this by a simple probabilistic analysis.

OUR CONTRIBUTIONS

An impressive result is that the driver can compute/update their shortest path result by receiving only a small fraction of the index. As such, huge communication cost will be spent on sending result paths on this model. 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.

SYSTEM ARCHITECTURE:

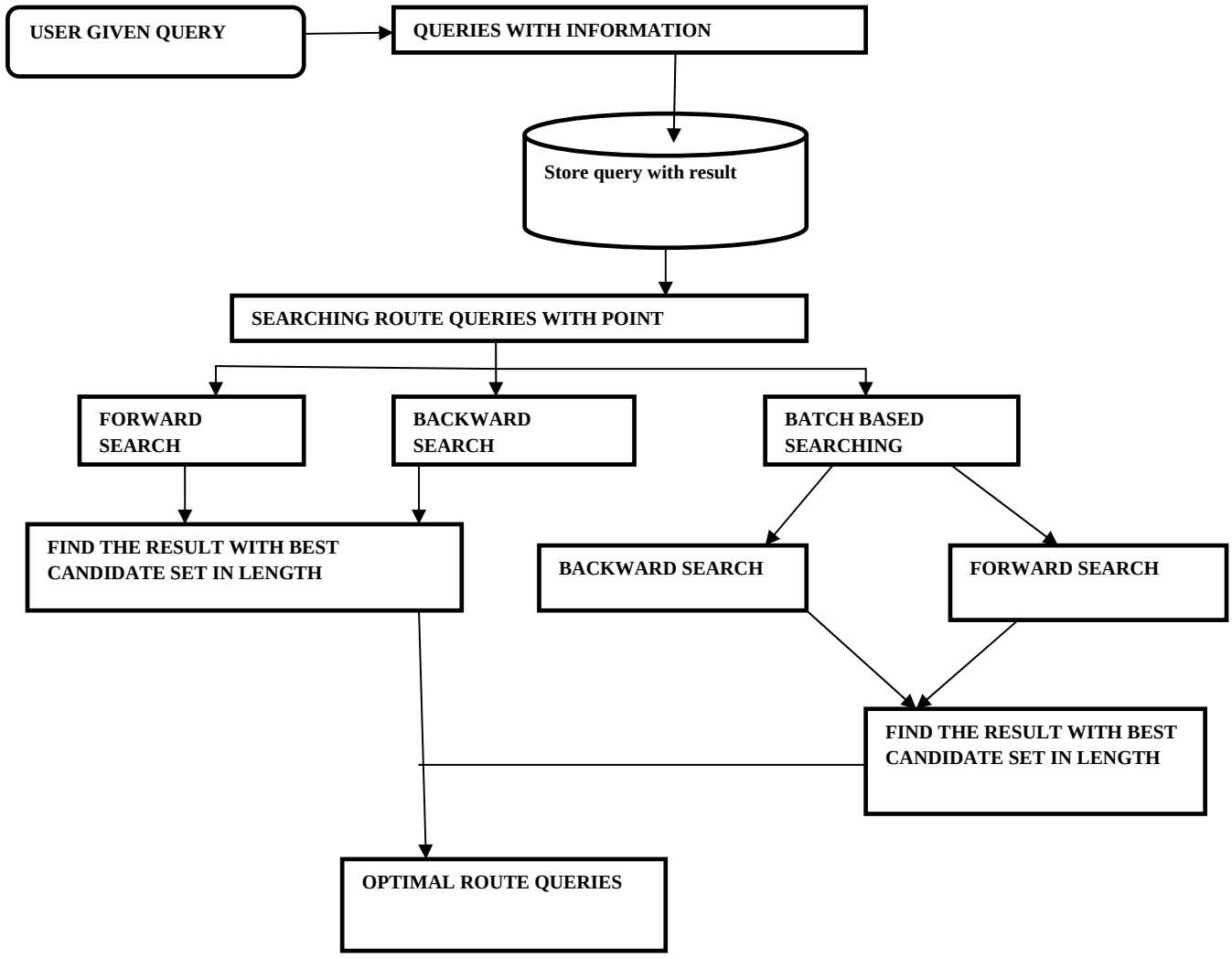

FIGURE 3

SENSOR NODE

The Sensor Node is used to sense all the nodes and the process in the network. Each and every node will be created via sensor. Sensor can also able to forward and receive the files and can also synchronize it successfully.

SEARCH ENGINE SERVER CONSTRUCTION

This module is an important module of our system. This module provides the ways to construct database of location information and other related information of that location

ROUTE QUERY CONSTRUCTION

This module facilitates our system to construct the user's query. This query construction is done for making efficient way for processing the query to give optimal result for the users.

VI ELLIPTIC BOUNDARY (EB) METHOD

Intuitively, in order to efficiently process shortest path queries we have to partition the road network into regions and use an index structure to guide the search through them. To satisfy the requirements, the index should be particularly concise, much more so than existing indexes designed for disk-resident networks. The Elliptic Boundary method (EB) follows this approach. Its crux is to first provide the client with an upper bound of the shortest path distance between vs and vt. This bound is used to prune (i.e., to avoid listening to) network information about nodes that lie too far away from vs and vt to affect the shortest path search. This is achieved by partitioning the network into regions, and placing in the index of EB information about the minimum and maximum possible distance from any partition to any other. EB owes its name to the fact that the search area is reminiscent of a network-based ellipse with foci the regions.

C) INDEX OF EB

The index of EB includes two components. The first defines partitions and provides a mapping of nodes into regions; the second specifies minimum/maximum distances between regions. To commence query processing, the client must first identify the regions Rs and Rt of the source vs and destination vt, respectively. This process is bound to the partitioning method used. A straightforward approach is to superimpose a Euclidean regular grid (of equi-sized rectangular cells) over the network, and consider the part of the network inside each cell as a region. In this approach the client could trivially map the Euclidean coordinates of vs and vt to regions Rs and Rt, requiring only knowledge of the grid granularity (e.g., k×m cell partitioning), and of the total spatial extent of the grid. The drawback of this approach is that some regions would contain too few nodes (or be empty), while others would be too full. This would reduce the benefits of partitioning and impede the search.

D) CLIENT-SIDE PROCESSING IN EB

Posed a query, the client tunes in the broadcast channel, and listens to the current packet. It retrieves the pointer to the next index and sleeps; it wakes up when the index starts being broadcast, and receives it in its entirety. Regions Rs and Rt are determined. The second component of the index (array A) is then used to derive an upper bound UB for the shortest path distance from vs to vt; the maximum value stored in ARi,Rj serves as UB. The correctness of this upper bound can be easily seen, since any path from source to destination has to pass through at least one border node of each Rs and Rt. The next step is to determine which regions must be received. Based again on A, the client needs to listen to only those regions R for which mindist(Rs, R)+mindist(R, Rt) ≤ UB, i.e., the sum of minimum distance from Rs to R plus the minimum distance from R to Rt is no larger than UB. Upon deciding which regions are necessary, it sleeps and wakes up when their data (contained nodes and adjacency lists thereof) are broadcast. When all necessary regions are broadcast and received, the client performs a Dijkstra search in their union (a sub-graph of the network) and reports the computed shortest path; this is guaranteed to be the correct answer in the entire network. Observe that access latency does not exceed one broadcast cycle.

VII NEXT REGION (NR) METHOD

While EB allows for selective tuning, its search space (i.e., the set of regions received by the client) may still be large. The problem is exacerbated when the source and destination are far away, as EB's network-based ellipse includes an increasing number of regions. In the extreme case where vs and vt are located in the furthest regions, it is possible that EB needs to receive all regions. Note that in this degenerate case, performance may actually be worse than Dijkstra's algorithm, since the broadcast cycle of EB is longer. In this section we present the Next Region method (NR), which avoids the above problem. The server again pre-computes the shortest paths between all border nodes of different regions, but now the index keeps for each pair of Ri, Rj the identifiers of intermediate regions appearing in any shortest path between any of their border nodes. These regions are guaranteed to contain the shortest path from any node in Ri to any node in Rj . To exemplify, consider that the index includes an n × n × n array A of boolean values, where n is the number of network regions. The bit in cell ARi,Rj ,Rk is 1 if and only if there exists a shortest path from Ri to Rj that traverses Rk. Given array A, the client knows in advance which regions are necessary for query processing. Consider the example in Figure 4. The source is in region R1, which has two border nodes; the destination is in R16, which has a single border node. There are two shortest paths between the border nodes of R1 and R16. One of them traverses regions R2, R6, R7, R11 and R12, and the other regions R5, R6, R7, R11 and R12. The NR index records that any shortest path from R1 to R16 may only pass through the union of the above region sets (shown gray in the figure), and the corresponding bits in A are set to 1. Array A as described above has n 3 size, and would lead to a large index. This, in turn, implies a long broadcast cycle, especially if the (1,m) scheme is applied. In Section 5.1 we show how we can retain the pruning effectiveness of NR while both (i) keeping the broadcast cycle short and (ii) achieving low access latency. We stress that the reduced access latency is provided by local, region-specific indexes, broadcast

immediately before the corresponding regions. This eliminates the need for (1,m) interleaving, and is fundamentally different from the common practice in the literature of having a global index, and replicating identical copies of it in the broadcast cycle.

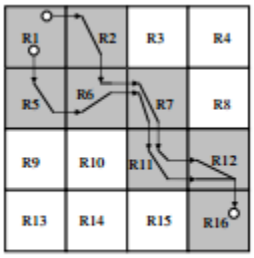

Figure 4: Shortest paths from R_1 to R_{16}

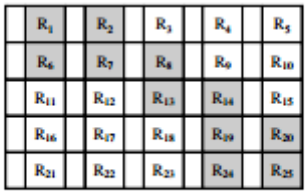

Figure 5: Needed regions

Which has a single border node? There are two shortest paths between the border nodes of R1 and R16. One of them traverses regions R2, R6, R7, R11 and R12, and the other regions R5, R6, R7, R11 and R12. The NR index records that any shortest path from R1 to R16 may only pass through the union of the above region sets (shown gray in the figure), and the corresponding bits in A are set to 1. Array A as described above has n 3 size, and would lead to a large index. This, in turn, implies a long broadcast cycle, especially if the (1,m) scheme is applied. In Section B we show how we can retain the pruning effectiveness of NR while both (i) keeping the broadcast cycle short and (ii) achieving low access latency. We stress that the reduced access latency is provided by local, region-specific indexes, broadcast immediately before the corresponding regions. This eliminates the need for (1,m) interleaving, and is fundamentally different from the common practice in the literature of having a global index, and replicating identical copies of it in the broadcast cycle.

E)INDEX OF NR

The first component of the index in NR is the same as EB, and is used to identify the source and destination regions Rs and Rt. Regarding the second component, the main idea is that, since the device will have to wake up every time a needed region is broadcast, it does not need to know all the required regions in advance. It suffices, instead, to only know when the next required region will be broadcast. When the client receives that region, it also listens to the adjacent local index in order to determine the next required region, and so on. This way, we keep the broadcast cycle small, we enable the client to receive only the relevant parts of (instead of the entire) indexing information, and allow the device to commence query processing shortly after tuning in for the first time, without employing the (1,m) scheme. Specifically, the index A m of region Rm is an array with n rows and n columns. A m is placed in the broadcast cycle immediately before Rm's data. Every cell A m Ri,Rj indicates the next region Rnxt in the broadcast cycle that is needed for a shortest path from Ri to Rj . Note that Rnxt could be Rm itself. Figure 5 shows the structure of the cycle, where the unlabeled slot before each region corresponds to its index.

F)CLIENT-SIDE PROCESSING IN NR

Posed a query, the device tunes in the channel, receives the current packet, and waits until the subsequent index is broadcast (for this to be possible, every packet in the cycle includes a pointer (offset) to the subsequent index). The client receives this index, and finds out what the next required region Rnxt is. It wakes up when Rnxt is broadcast and keeps listening until A nxt+1 is also received. From A nxt+1, it determines the next needed region, and so on. Note that if the end of the current broadcast cycle is reached, another starts, and processing continues as if it was the same cycle. When the latest index received indicates that Rnxt is a region that the client already possesses, listening stops and a Dijkstra search computes the shortest path over all collected regions. Similarly to EB, the access latency in NR does not exceed one broadcast cycle. Regarding tuning time and memory requirements, we expect NR to be superior to EB, as the client listens only to a subset of the regions necessary in EB. The same holds for CPU time at the client. Pre-computation cost is identical to EB (assuming the same partitioning), as the same shortest paths among border nodes are computed. To illustrate, consider the broadcast cycle in Figure 5. The user wants to find the shortest path from a source in R1 to a destination in R25. The needed regions for this shortest path computation are shown in gray color, but the client does not know this in advance. Assume that the query is posed while R11 data are broadcast, which points the client to index A 12. Index A 12 (shown in Figure 6) indicates that R13 is the next needed region, so the device sleeps and wakes up to receive R13 and also the adjacent index A 14 . A 14 indicates that R14 is also required, so the client continues to receive data from the channel, until A 15 points to R19, as shown in Figure 6. The device sleeps until R19 is broadcast, and so on. The process continues this way until R8 is received and index A 9 points to the already available R13; listening stops and the shortest path is computed. Algorithm 2 in Appendix B formalizes this process.

So far we have assumed that source and destination are network nodes. In practice this may not always be the case, i.e., the source/destination could be at arbitrary locations on the network. EB and NR work as described, the difference being that the border nodes of a region are now defined as the intersections of its network edges (i.e., with the boundary of the region).

VIII QUERY PROCESSING:

Distance signature is superior to the existing indexes in terms of the diversity of the kinds of queries supported. Since it indexes the underlying distances, rather than the solution for a particular type of queries, it can be applied to virtually any queries relating to distances. In this section, we present the algorithms to process common spatial queries based on the distance signatures. We discuss range

queries, and generalize the processing paradigm to other query types such as aggregation queries and network joins.

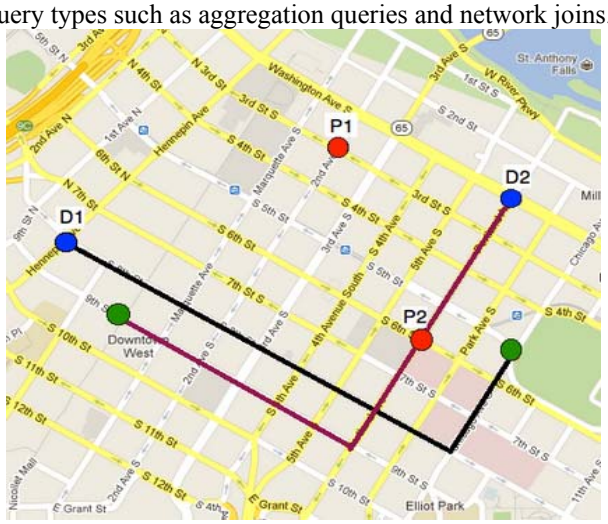

FIGURE 6

G) DISTANCE SPECTRUM PARTITION:

A good partition of distance spectrum must consider the following factors:

DATASET DISTRIBUTION

The distribution, especially the density of the dataset, determines the object distribution in the distance spectrum. Obviously, a dense dataset requires more categories than a sparse dataset does.

QUERY LOAD

For example, the distance threshold ϱ of a range query, query affect how precisely the distance spectrum should be partitioned. In order to quantify the query load, we define "spreading" (denoted by sp) as the distance threshold of those objects that are interesting to the query. For range queries, sp = ϱ, sp is the distance of the k+1th nearest neighbor. Obviously, the distribution of sp should affect the partition of distance spectrum so that the signatures can achieve maximum performance.

STORAGE AVAILABILITY

Accurate partition requires more storage to encode the categories than coarse partition. As such, the availability of disk storage is also a concern.

IX CONCLUSIONS

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, an impressive result is that the driver can compute/update their shortest path result by receiving only a small fraction of the index. As such, huge communication cost will be spent on sending result paths on this model. We extended our solution on time dependent networks. Since the decision of a shortest path depends not only on current traffic data but also based on the predicted traffic circumstances.

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).

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

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

[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 Algorithmic and Combinatory (ALENEX/ANALC), pp. 100-111,

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

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

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

[8]. F. Zhan and C. Noon, "*Shortest Path Algorithms: An Evaluation Using Real Road*

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

[10]. 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,.

[11] . 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.

[12]. G. Kellaris and K. Mouratidis, "*Shortest Path Computation on Air Indexes*," Proc. VLDB Endowment, vol. 3, no. 1, pp. 741-757.

[13]. E. W. Dijkstra. *A note on two problems in connexion with graphs. Numerische Mathemat*ik, 1:269–271.

[14]. P. Hart, N. Nilsson, and B. Raphael. *A formal basis for the heuristic determination of minimum cost paths.* IEEE TSSC, 4(2):100–107.

[15] E. K¨ohler, R. H. M¨ohring, and H. Schilling. *Fast point-to-point shortest path computations with arc-flags. In 9th DIMACS Implementation Challenge - Shortest Paths*, 2007.

[16] G. Kellaris and K. Mouratidis, "*Shortest Path Computation on Air Indexes," Proc. VLDB Endowment,* vol. 3, no. 1, pp. 741-757.

[17]. Schulz, F., Wagner, D., Weihe, K.: *Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport.* ACM J. of Exp. Algorithmics.

[18] . Wagner, D., Willhalm, T., Zaroliagis, *C.: Geometric Containers for Efficient Shortest-Path Computation.* ACM J. of Exp. Algorithmics.

[19]. Holzer, M., Schulz, F., Wagner, *D.: Engineering Multi-Level Overlay Graphs for Shortest-Path Queries*. In: Proceedings of the 8th Workshop on Algorithm Engineering and Experiments.

[20] Holzer, M., Schulz, F., Wagner, D., Willhalm, T.: *Combining Speed-up Techniques for Shortest-Path Computations*. ACM J. of Exp. Algorithmics.