Volume 3, Issue 8, August 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Fast Index Maintenance Along with Secondary Indexes

V. Sujitha
Scholar/Department of Computer Science & engineering, JNTUA/ Madanapalle Institute of Technology & Science, Angallu,Madanapalle-517325,AP,India,

K. R.K.Satheesh
Associate professor /Department of Computer Science &engineering, JNTUA/ Madanapalle Institute of Technology & Science,Angallu,Madanapalle-517325,AP,India,

***Abstract* —Point of interest (POI) sequences also called as route collections obtained from GPS/GIS enabled devices helps commuters reach their destinations faster by using the sorted routes information obtained from the route collections. This sorting process involves path query evaluation on large disk resident route collections that are regularly updated. Updates involve additions and deletions of routes. Previously graph based procedures offered sorted solutions, they had huge computation overhead. So we propose to use generic search-based paradigms that exploit transitivity information within the routes, and differ in their expansion phase using fast index maintenance procedures such as Link Traversal Search (LTS-k) algorithms along with maintaining secondary indexes for faster retrieval process. The proposed system offers better performance driven aspects leveraging on reduced computations and a practical implementation validates the claim.**

***Key Words*—Secondary index, indexing, updated route collections, interesting points.**

I. Introduction

A geographic information system (GIS) integrate hardware, software, and information for capturing, organizing, analyzing, and displaying all forms of geographically referenced information.GIS allows us to view, understand, query, infer, and visualize data in many ways that reveal associations, patterns, and trends in the form of maps, globes, reports, and charts.GPS technology is one such practical implementation of GIS systems that provides routing systems, GPS tracking devices, GPS surveying and GPS mapping. GPS itself does not provide any functionality beyond being able to receive satellite signals and calculate position information. As a first example, believe people who visit Athens and use GPS-enabled devices to track their points of interests. At the end of each day or after they return to home, they create routes through interestingplaces they visited, either manually, or employing works. Figure 1 shows two touristic routes in Athens.The First, r1, starts on or after the National Technical University of Athens and ends at the Museum of Acropolis. The second, r2, starts on or after the Square and ends at the Acropolis Entrance.Web sites maintain aimmense collection of routes, like the above, with PointOf Interests from all over the world. These collections are regularlyentry updated as users continuously share new interesting routes.Point of interest (POI) sequences also called as route collections obtained from GPS/GIS enabled devices helps commuters reach their destinations faster by using the sorted routes information obtained from the route collections. This sorting process involves path query evaluation on large disk resident route collections that are regularly updated. Updates involve additions and deletions of routes. A route collection can be trivially transformed to a graph; hence, path queries can be evaluated using regular graph search techniques. Such methods follow any one of two paradigms.

- The first employs graph traversal methods, such as depth first search (DFS).
- The second compresses the graph’s transitive closure, which contains attain ability information, i.e., whether a path exists among several pair of nodes.

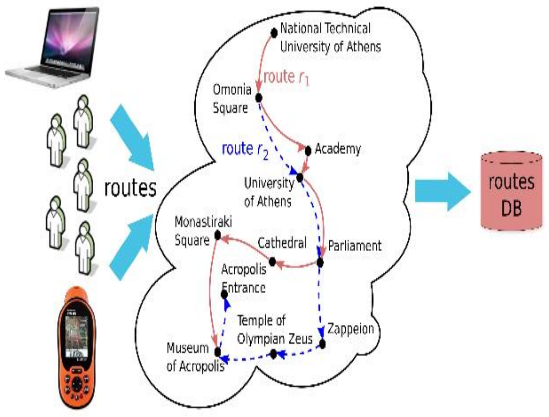

Figure 1: Two touristic routes in the city of Athens

Both paradigms share their strengths and weaknesses. While transitive closure techniques are the fastest, they are generally suitable for data sets that are not regularly updated, or when the updates are localized, since they need expensive precomputation. On the other hand, DFS are simplysustainable, but are slow as they may visit a large part of the graph.So a better system is required that produces better sorted route collections over dynamic data with reduced computations.

To solve above problems, two generic search-based paradigms are proposed that exploit transitivity information within the routes, and differ in their expansion phase [1]. For each route that contains the current search node, route traversal search (RTS) expands the search considering all successor nodes in the route, while link traversal search (LTS) considers only the next link. Both paradigmsthat use fast index maintenance procedures terminate when they reach a route that leads to the target, and are faster than conventional search.RTS employs an inverted file method, R-Index, on the nodes of the route collection. Route Traversal Searchalong with Transitions (RTST) uses the T -Index (in addition to R-Index) that captures transitions among routes allowing earlier termination.

In this paper, we propose to use generic search-based paradigms that exploit transitivity information within the routes, and differ in their expansion phase using fast index maintenance procedures such as Link Traversal Search (LTS-k) algorithms along with maintaining secondary indexes for faster retrieval process. LTS employs an augmented variant of R-Index and features a similar termination condition to RTS. Similarly, Link Traversal Search with Transitions (LTST) has a stronger condition based on the T -Index. The LTS-k algorithm forgoes the high storage and maintenance cost of T -Index and features a tunable termination condition, which is at least as strong as that of LTS and can become as strong as that of LTST. The proposed system offers better performance driven aspects leveraging on reduced computations and a practical implementation validates the claim.

For performance improvement we suggest to use fast index maintenance procedures along with Secondary indexes.In table-based partitioning, an index that is not a partitioning index is a secondary index. A secondary index can be nonpartitioned or partitioned. You can construct an index on a table to implement a uniqueness constraint, to cluster data, or to make available access paths to data for queries.Secondary Indexes provide another path to access data. Let's say that you were developing a road trip to a place. To determine the best way to get there, you need to employ a map. This map will give you many alternatives to plan your trip. In this case, you need to get there, as soon as possible. So you choose the best route to get there in the shortest period of time. Secondary indexes work related to this above example because they provide another path to the data. Secondary Indexes (when defined) do take up additional space but they improve speeds and aids in faster retrieval process.

II. Relatedwork

Cohen et al. [2] propose 2-hop labels.They categorize a subset of the nodes that best capture the reachability informationof a graph. Thus, for each nodev, they construct a list with part of the nodesthat can reachv(Lin[v]) and another one with part of the nodes reachablefromLout[v]).This method requiresO(|V|.√|E|) space and can determinethe existence of a path between two nodesvs,$v_t$by checking whetherLout[v_s]andLin[v_t] have a common node, the so called centerv_{center}. To recognizethepath from$v_s$tov_t, we need to repeat the process for the paths from$v_s$to$v_c$and from$v_c$tov_t. The problem with this advance lies in themanufacture cost.

Jin et al. [3] propose the 3-hop indexing application. The basic idea is to use a sequence of nodes, instead of a single node, to encode the reachability information. Agrawal et al. [4] propose a labeling scheme that assigns to each node a sequence of intervals, based on the postorder traversing of graph's spanning tree. Updates are handled by leaving gaps in postorder numbers. Although not discussed, PATH queries can be answered on the DAG by computing the ancestors of the target node.

In the context of labeling schemes for graphs, [4] proposes an interval labeling scheme. Considering both the spanning tree of the graph, and the remaining edges, they assign to each nodeva sequence of intervalsL[v]. In [5],Wang et al. introduce Dual-Labeling for sparse graphs. In [6], Trißl et al. introduce GRIPP scheme for large graphs. The idea in [7] is instead ofconstructing the spanning tree of the graph, to partition the graph into a set of paths Pand then create the so called path-tree coverG[T]. The path-treecover is a graph formed by the paths ofP(as nodes) and the edges of the initialgraph that are not included in any path.

HOPI [8], [9] reduces the construction time of 2-hop by exploiting graph partitioning, which works well for forests with a small amount of connections between the different subgraphs, e.g., collections of XML documents. Updates are handled by applying the construction method of [8]. HOPI is able to find elements, e.g., book, author, in an XML document that match XPath expressions, e.g., //book//citation//author (where "//" is the ancestor-descendant operator). However, the focus of the work is to identify these elements and not detect the full path on the XML documents that contains them.

III. Background Work

A. *Route Traversal Search*

The Route Traversal Search algorithm has the following key features. First, node traversing is done which is similar to depth-first search. However, when increasing the current search node n_q, all successor nodes for each route are considered by RTS that includes n_q. Second, currently termination check is employed, according to the reachability information within the routes, to significantly shorten the search. Both standards depend on the inverted file R-Index on the route collection which associates a node with the routes that contain it. The RTS algorithm takes as inputs: R-Index, a route collection R, the source n_s and target node $n_t$and returns a path from n_s to n_t, if one exists, or null otherwise. RTS proceeds similarly to depth-first search. Initially, the stack Q and H contain source node n_s, while A is empty. RTS proceeds iteratively popping a single node n_q from Q at each iteration. The algorithm terminates when there exists a route rc that contains both n_q and target n_t, that n_t comes after n_q. The procedure is similar to a merge join, as both routes(n_q) and routes(n_t) lists are sorted by the route identifier that finishes when a common route rc is found.

B. Link Traversal Search

Link Traversal Search perform fewer iterations than conventional depth-first search on the route collection graph GR, they share three shortcomings. First, they carry out redundant iterations by visiting nonlinks. To understand this, consider that the current search node n_q is not a link and belongs to a single route r_i. Further, assume that the algorithm has visited n‘, which is the link directly before n_q. Observe that if the termination condition does not hold at n‘, then it neither holds at n_q. To create matters poorer, retrieving routes ($n_{q)}$ is pointless as it contains a single route ri in which all nodes after nq are already in the stack. The second shortcoming is that the termination check is expensive. For current search node n_q, bring to mind that both RTS retrieve lists routes(n_q) and routes(n_t) from R-Index. This cost is amplified by the number of iterations, as the algorithms achieve the check for every node popped. The final shortcoming is due to the traversal policy. For each route that the current search node belongs to, the algorithms place in into the stack route subsequences thatcontain a very large number of nodes. This increases the space requirements of Q.

IV. Proposed System

The Link Traversal Search with Transitions algorithm enforces a stronger termination check than LTS using the transition graph of the route collection. In particular, the LTST algorithm, similar to RTST, finishes when it reaches a node that is closer than two routes away from the target. To achieve this, LTST uses information from the T –Index. The transition graph is stored in a modified adjacency list representation denoted as T -Index.

A. The LTS-k Algorithm

The LTST algorithm terminates as soon as the current search node is within two routes from the target. This strong termination condition is achievable due to the information stored in T -Index. However, the size of T -Index is quadratic with respect to the number of routes, which makes it impractical for large collections.The LTS-k algorithm that operates without the T -Index, and features a tunable termination condition based on parameter k. Particularly, LTS-k stops when it reaches a route $r_i$that leads via link n‘to a route r_j containing the target n_t, with the requirement that n‘ isat most k links earlier thann$_t$ in r_j. Note that when k is set to 0, the algorithm reduces to LTS. On the other hand, for aadequately high k value (larger than the maximum number of links in any route), LTS-k terminates when it visits a node that is less than two routes from n_t, exactly like LTST. In this case, however, LTS-k spends more time compiling the target list compared to LTST, since the latter has access to T -Index that materializes the transition information betweenany two routes.

In the first phase, LTS-k constructs a list Lwith all links that are within k links from the target in someroute, including $n_t$itself. To locate these nodes, thealgorithm retrieves all routes that contain n_t andinserts into L the k links before n_t (if they exist) in each route. An entry of L has the form (n‘,ri:o‘i) which means that link n‘lies in the same route ri with targetn$_t$ and is within k links away from it. Note that even though alink in L may appear in several routes, LTS-k only keeps asingle entry per link. Second, L is not the set of all links that are withink links from the target. Rather, L contains a subset of onlythose links that are in the same route with n_t, first and foremost forefficiency reasons. In order to reach all links within k linksfrom n_t, the algorithm would need to perform a breadthfirstsearch starting from n_t following the reverse edges ofthe conceptual reduced routes graph.

B. Updating Route Collections

We consider route collections thatdo not fit in main memory and thus, all indices are stored as inverted files on secondary storageand maintained by batch updates. Inverted files are moreefficient when their lists are stored in a contiguous way.In our work, we adopt the second strategy i.e., fast index maintenance procedures along with Secondary indexes. In table-based partitioning, an index that is not a partitioning index is a secondary index. A secondary index can be nonpartitionedor partitioned. You can construct an index on a table to implement a uniqueness constraint, to cluster data, or to make available access paths to data for queries. Secondary Indexes provide another path to access data. Secondary Indexes (when defined) do take up additional space but they improve speeds and aids in faster retrieval process.

When we dealing with each new route separately, it is notan efficient method for updating the collection.A commonsolution is to build inverted indices in memory considering allthe new routes and to exploit them for evaluating the queriesin parallel with the disk-based indices. Each time a set ofnew routes arrives, only the memory-based indexes are updatedwith minimum cost. Then, to return the changes in the disk-based indexes, there are three possible strategies: (a) rebuildingthem from scratch using both the old and the new routes, (b)merging them with the memory resident ones and (c) lazilyupdating index lists when they are retrieved from disk duringquery evaluation.

To handle frequent updates,we performlazy updates, deferringpropagation of changes to the disk by maintaining additional information in mainmemory. Then, at some time, a batch update process reflects all changes to the disk resident indices. Insertionsarehandled bymergingmemory-resident informationwith disk-based indices, while deletions requirere building of the affected lists.

1) *Insertions:* To support lazy updating for an insertion, we maintain a main memory list for eachdisk resident list affected. The main memory lists contains two types of entries. Anentry prefixed with the + symbol is new and must be added to the disk-based list.An entry prefixed with the±symbol exists on disk but must be updated.Indices are updated in two phases.

- Buffering updates the memory resident lists and occurs online every time a new route is inserted in the collection.

- Flushing propagates every changes to the disk-based indexes and is thus executed periodically offline.

When retrieving a disk-based list: (1) all main memory (+ and ±) entries are also considered, and (2) all disk-based entries that have acorresponding±main memory entries are ignored.

2) *Deletions:*Deletions need different treatment compared to insertions, as many entries acrossmultiple lists are affected. Identifying them would require a large number of diskaccesses. Therefore, a buffering phase does not occur when a route deletion arrives.Rather, a list is maintained that contains all routes deleted since thelast flushing.Then, during the execution of a search, retrieved entries that contain a deleted routeare simply discarded. This design choice may influence the performance of the linktraversal search algorithms, mainly because the demotion of a link to a non-linknode is not captured and thus non-link nodes may be visited. However, the deletionof a node is captured and hence thecorrectness of all proposed system is not affected.

V. Performance

As varying the links/nodes ratio increases the link frequency decreases and finding a path becomes more difficult. The target list of all methods decreases with |N|, and the initialization cost of LTST and LTS-k decreases. Correspondingly,the total execution time increases for DFS and LTS, while it first decreases and ultimately increases for LTST and LTS-k.

As |N| increases, even though the number of links increases, each of them is contained in fewer routes. Therefore, the reduced routes graph GR becomes sparser, which means that finding a path becomes harder. Subsequently, the initialization cost of LTST and LTS-k decreases with |N|.

Let’s say that you were planning a road trip to a place. To determine the best way to get there, you need to employ a map. This map will give you many alternatives to preparation your trip. In this case, you need to get there, ASAP. So you choose the finest route to get there in the shortest period of time. Secondary indexes work related to this above example because they provide another path to the data.

VI. Conclusion

A geographic information system (GIS) integrates hardware, software, and information for capturing, organizing, analyzing, and displaying all forms of geographically referenced information. In this paper, we propose to use generic search-based paradigms that exploit transitivity information within the routes, and differ in their expansion phase using fast index maintenance procedures such as Link Traversal Search (LTS-k) algorithms along with maintaining secondary indexes for faster retrieval process. LTS employs an augmented variant of R-Index and features a similar termination condition to RTS. Similarly, Link Traversal Search with Transitions (LTST) has a stronger condition based on the T -Index. The LTS-k algorithm forgoes the high storage and maintenance cost of T -Index and features a tunable termination condition, which is at least as strong as that of LTS and can become as strong as that of LTST. The proposed system offers better performance driven aspects leveraging on reduced computations and a practical implementation validates the claim.

References

[1] P. Bouros, S. Skiadopoulos, T. Dalamagas, D. Sacharidis, and T.K.Sellis,“Evaluating Reachability Queries over Path Collections,”Proc. Int’l Conf. Scientific and Statistical Database Management (SSDBM), pp. 398-416, 2009.

[2] Cohen, E., Halperin, E., Kaplan, H., Zwick, U.: Reachability and distance queriesvia 2-hop labels. In: SODA. (2002) 937–946.

[3] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry, “3-Hop: A High- Compression Indexing Scheme for Reachability Query,” Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 813-826, 2009.

[4] R. Agrawal, A. Borgida, and H.V. Jagadish, “Efficient Management of Transitive Relationships in Large Data and Knowledge Bases,” Proc. ACM SIGMOD Int’l Conf. Management of Data, pp. 253-262, 1989.

[5] H. Wang, H. He, J. Yang, P.S. Yu, and J.X. Yu, “Dual Labeling: Answering Graph Reachability Queries in Constant Time,” Proc. Int’l Conf. Data Eng. (ICDE), p. 75, 2006.

[6] Trißl, S., Leser, U.: Fast and practical indexing and querying of very large graphs.In: SIGMOD Conference. (2007) 845–856.

[7] Jin, R., Xiang, Y., Ruan, N., Wng, H.: Efficiently answering reachability querieson very large directed graphs. In: SIGMOD Conference. (2008) 595–608.

[8] R. Schenkel, A. Theobald, and G. Weikum, “Hopi: An Efficient Connection Index for Complex xml Document Collections,” Proc. Int’l Conf. Extending Database Technology (EDBT), pp. 237-255, 2004.

[9] R. Schenkel, A. Theobald, and G. Weikum, “Efficient Creation and Incremental Maintenance of the Hopi Index for Complex XML Document Collections, ”Proc. Int’ lConf . Data Eng.(ICDE),pp.360-371, 2005