

Efficient Dynamic Traffic Navigation with Hierarchical Aggregation Tree

Yun Bai¹, Yanyan Guo¹, Xiaofeng Meng¹, Tao Wan², Karine Zeitouni²

¹ Information School, Renmin University of China
100872 Beijing, China
{guoyy2, xfmeng}@ruc.edu.cn

² PRISM Laboratory, Versailles University,
45, Avenue des Etats unis
78000 Versailles, France
{firstname.surname}@prism.uvsq.fr

Abstract. Nowadays, the rapid advances in wireless communications, positioning techniques and mobile devices enable location based service such as dynamic traffic navigation. Yet, it is a challenge due to the highly variable traffic state and the requirement of fast, on-line computations. This motivates us to design an intelligent city traffic control system, which is based on an extension of the Quad-tree access method that adapts better to the road networks while it maintains aggregated information on traffic density according to hierarchy levels. Based on this index, a view-based hierarchy search method is proposed to accelerate the optimal path computation. At the end, the experiment will show this index structure to be effective and efficient.

1 Introduction

These advanced technologies make it possible for a vehicle (here referred to as a moving object) to have sophisticated onboard wireless equipment installed at a reasonable price [1]. On this condition, location-based services are becoming more and more popular. Traffic navigation service, qua one of this kind of services, receives the special attention because of the closed relation with modern life. Besides, traffic surveillance technologies (either vehicle based using transponders, GPS or cell phones etc) allow monitoring and broadcasting of traffic conditions in real time. This encourages some promising applications that monitor cities' traffic, and allow new services to consumers such as dynamic traffic navigation service. It is believed that the dynamic traffic navigation will become widely spread because it can provide exact useful information on driver's current position, optimal path to destination, traffic congestion and so on and so forth. Yet, it is a challenge due to the highly variable traffic state and the requirement of fast, on-line computations. In research side, certain work, such as [2], uses the technique of prediction which forecasts potential congestions and thus calculates the optimal path for each moving object. It needs each moving object to provide its start time, start location and its destination. However, obtaining all the information of each object is unrealistic, and traffic conditions are difficult to forecast (e.g. traffic jams produced by any unpredictable bursting event such as an accident). Therefore, dynamic navigation techniques can not be totally based upon prediction models. In other words, efficient dynamic navigation is a challenge due to the highly variable traffic conditions and the requirement of fast, on-line computations.

This motivates us to design an intelligent city traffic control system, providing the user, always in real time and in a continuous fashion, the optimal path to destination considering traffic conditions. Although the optimal path finding problems is one of the most fundamental problems in transportation networks analysis [3, 4], here we will not attempt to propose an algorithm to solve another variation of path finding problems. Instead, we have adapted the most efficient Dijkstra algorithm [5] for real time shortest path calculating in order to answer path finding request in a continuous fashion. In this paper, we propose at first a novel indexing method, namely HAT, combining spatial indexing technology and pre-aggregation for traffic measures.

On one hand, this spatial index is more suitable to road networks, and could support other kinds of location-based services especially on road network; on the other hand, the aggregated information stored in each hierarchy level may filter congested zones in advance. Then, based on this index structure, a notion of personal “view” is defined. First, it improves the performances of optimal path finding by limiting search to regions that are likely to be crossed after the current user’s location. Second, it optimizes the local memory occupancy. Finally, we derive a navigation algorithm and a Dynamic Navigation System, named *DynSA*, and prove, through extensive experiments, its efficiency and effectiveness.

The paper is organized as follows: Section 2 describes, in detail, a novel road index HAT; Based on this index structure, section 3 depicts an efficient method for searching optimal path within the person “view”; Section 4 details the architecture and experimentation of the system. Finally, summary is presented in Section 5.

2 Hierarchy Aggregation Tree (HAT)

Navigation intensively uses the retrieval of network sections in which objects move. So, an efficient spatial index is then a key issue. A road is usually represented as a line string in a 2-dimensional space. Spatial access methods, such as R-tree, PMR Quad-tree [6] and Grid file [7], can be consequently adopted for networks’ indexing. However, R-tree indexing structure produces large overlapping MBRs (Minimum Bounding Rectangle), which makes the search inefficient. Furthermore, originally tailored for indexing rectangles, applying R-tree to a network will result in large amount of dead space indexing (the gap between a linear feature and its MBR). Although PMR Quadtree and Grid File have not the overlapping problems, the uniform geometric partition do not adapt to non-uniform road distribution in space. The cell number in Grid File increases while PMR Quadtree becomes more and more unbalanced, which affects the search efficiency. Since the R-tree, PMR Quadtree and Grid File are not suitable for optimal path searching in traffic environments, to improve the efficiency of query processing, we put forward a new indexing method named Hierarchy Aggregation Tree (HAT). It is based on two structures: road and region. The former is a segment of road which has not intersection with other road except two extremities. The latter is similar to a MBR, in addition, it contains a supplement information which stores an aggregated value over it. The principal functionality of this aggregated information is to filter the regions having a high traffic density on the same hierarchy level.

2.1 Basic Idea

HAT is set up based on the spatial information of roads. Unlike R-tree, HAT references, edges and nodes so that it avoids dead space. Moreover, because node MBRs do not overlap, the search is more efficient. Basically, this method is inspired from PMR Quadtree’s non-overlapping index, except that the space partition in HAT may be skewed and the resulting tree will be balanced. Furthermore, for each region, HAT stores additional information on traffic density at different granularity levels. As shown later, this brings a filter capability for path finding process.

2.2 Index Building

HAT is constructed by partitioning the index space recursively. The space is divided according to the distribution of network segments, by an adaptive and recursive split of space in four sub-regions. When the amount of roads, namely capacity in a leaf node N exceeds B (B is the predefined threshold for split), N is to be split and the corresponding region is to be partitioned into four sub-regions. The split method of HAT satisfies the following two rules:

Rule1: Capacities in four sub-regions should almost be the same. Suppose Max is the maximum capacity and Min is the minimal capacity of four sub-regions. Then the difference between Max and

Min should not exceed predefined proportion $P1$, which is expressed by the following inequality: $(Max-Min) \leq Max \times P1$ \lceil Symbol \rceil denotes the minimal integer more than the value in it.

Rule2: The sub-regions crossed by a road should be as few as possible, namely the copies of entries for the road should be as few as possible. Suppose S is the sum of four capacities, C is the capacity of original region. S should be as close to C as possible, i.e. the difference between S and C should not exceed the predefined proportion $P2$, which is expressed by the following inequality: $(S-C) \leq S \times P2$

The main idea of *Split* algorithm is to firstly partition the region into four ones of equal size; then no more modification is needed if the result satisfies *Rule1* and *Rule2*. Otherwise the partition point is adjusted. Let N be an overflowing node, and let R be its corresponding region of capacity $C(C > B)$. To process *split* operation, R is partitioned at a partition point (x_s, y_s) into four equal sub-regions R_1, R_2, R_3, R_4 . (x_s, y_s) is initially set to (m_{median}, y_{median}) , i.e. the median point among road coordinates in R , then, it should be adjusted to fulfil the above rules. To do so, the split axis is pushed so that it minimizes segment split, as sketched in Fig.1 and Fig.2.

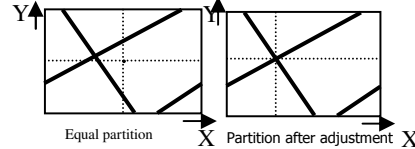
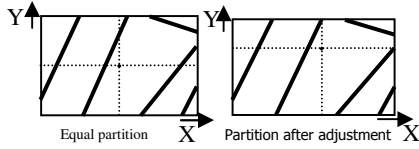


Fig. 1. Adjusting partition for a region without crossings **Fig. 2.** Adjusting partition for a region with crossings

By the above method, although the partition of index space is skewed, the resulting tree is balanced and the copies of roads are reduced which save the occupancy of index and the searching efficiency.

3 Dynamic Navigation Query Processing

Dynamic navigation query refers to finding a path through which a user will take least time, distance or cost to reach the destination. In this section, we introduce a new method for path searching, using the available moving objects stream aggregation and HAT index.

3.1 Optimal Path Search based on View

We notice that in real world, users do not prefer detour. Thus their travel only involves parts of the whole map. Considering this, referring to the concept of “view” in relational databases, we carry out the view-based searching method. When finding the optimal path, only consider the area from the user’s current position to his/her destination, which is referred to as “reference area” shown in Fig.3. The resulting path is then found from the set of those roads crossing the reference area, which are referred to as “candidate roads”.

Reference area is a logic notion. To retrieve all the candidate roads in HAT, firstly the algorithm retrieves those underlying regions contained in or intersects with the reference area. This spatial union of underlying regions is then referred to as “searching area” which is a partial view of the whole map shown in the right figure.

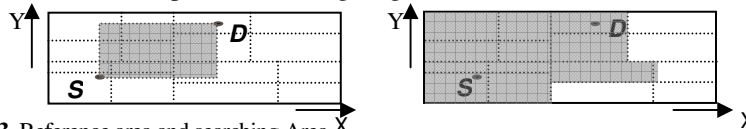


Fig. 3. Reference area and searching Area

Since HAT corresponds to the whole map, the “view” corresponds to some nodes of HAT. These nodes remain the tree-like structure referred to as “view tree”. Consequently, not accessing the whole HAT but only parts of it will take less time implementing the dynamic navigation task.

In addition, with the user becoming closer and closer to the destination, the size of the view tree will surely reduce, and release memory resources. Processing on such a smaller and smaller view tree will surely enhance the efficiency.

3.2 Hierarchy Search

Based on the structure of view tree, we adopt a method similar to “drill-down” process in OLAP, named “hierarchy search”. The hierarchy search is a top down process that always chooses the region of better traffic conditions as the mid region to go through until it finds the resulting path. Considering a movement from S to D , the user may go through different regions along different paths.

Actually, several cases could be distinguished. The first is when S and D belong to the same finest region, which means they are in the same leaf-node. In this case, a direct call to Dijkstra search suffices. The second case is when S and D belong to the same region, but at a higher level of the tree. We then continue to descendant nodes till S and D will appear in different sub-regions or in a leaf-node. In the other cases, when S and D belong to two different regions (denoted as R_s and R_d), we should find mid regions from S to D by selecting a sub-region (denoted as R_m) providing the rough direction of $R_s \rightarrow R_m \rightarrow R_d$. Since each region is partitioned into four ones, R_s and R_d are either adjacent to each others, or separated by one region of the same level as shown in Fig.4. If R_s and R_d are adjacent, the algorithm makes recursive calls to *HierarchySearch*, starting from their corresponding nodes. Whereas, if they are not adjacent, recursive calls will include one of the two sub-regions according to its aggregate value. For instance, in the right of Fig.4, suppose the density in R_3 is smaller, then the determined direction will be $R_s \rightarrow R_3 \rightarrow R_d$ as the arrow shows.

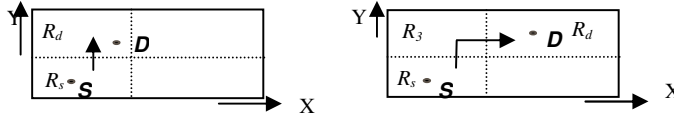


Fig. 4. Different distribution of S and D

Notice that the path from one region R_s to an adjacent region R_d will necessarily pass through a road that crosses their frontier. Suppose r_1, \dots, r_m are m such roads spanning R_s and R_d . Therefore, $R_s \rightarrow R_d$ can be concretely transformed to m selections: $S \rightarrow (r_i.x_1, r_i.y_1) \rightarrow (r_i.x_2, r_i.y_2) \rightarrow D (1 \leq i \leq m)$. The endpoint $(r_i.x_1, r_i.y_1)$ is in R_s and the other endpoint $(r_i.x_2, r_i.y_2)$ is in R_d . Each selection results in different paths (denoted as $path_i$). For each selection, we issue the same hierarchy search under node N to find the optimal path (denoted as $path_i.sub-path_1$) from S to $(r_i.x_1, r_i.y_1)$ and the optimal path (denoted as $path_i.sub-path_2$) from $(r_i.x_2, r_i.y_2)$ to D . Then, we join $path_i.sub-path_1$ and $path_i.sub-path_2$ by r_i and get the entire $path_i$. Finally, we select the one of minimal cost among every $path_i$ as the final optimal path from S to D .

4 Performance Evaluation

4.1 The Architecture of DyNSA System

Adopting the aforementioned index structure and navigation method, we have designed and implemented an intelligent city traffic control system, named *DyNSA* (Dynamic Navigation System based on moving objects stream Aggregation), which aims at providing high quality of dynamic navigation services for Beijing Olympics in 2008. An overview of this system architecture is shown in Fig.5.

This system consists of multiple managers: *TIR* (Traffic Information Receiver), *TIM* (Traffic Information Manager) and *Query Processor*. *TIR* is an information receiver which continuously sends traffic information to *TIM*, it can be considered like current TMC which captures real

traffic information. In *TIM*, aggregated information of each road segment is timely refreshed according to current traffic information and the region aggregation on each HAT's hierarchy level is thus recalculated. *Query Process* is in charge of users' navigation requests. When a navigation request is coming, it will send it to a *View Manager*, and then a corresponding view tree on HAT will be created. The *Service Agent* will perform the view based hierarchy searching on it, and finally, the optimal path will return to the user. Since the *View Manager* keeps a consistency between the view tree and the HAT, a recalculation will happen if necessary and will be sent to the user until she/he arrives to her/his destination. The underlying index structure of *RIM* and *Query Processor* are both based on HAT.



Fig. 5. System Architecture of *DyNSA*

5.2 Performance Studies

DyNSA was implemented in Java, so that both PMR Quadtree and Grid index were also implemented respectively in the same platform. For the experimental data, we have adopted several real road network datasets on which some moving object datasets are generated by Brinkhoff's generator [8]. The parameters used are summarized in Table 1, where values in bold denote default used values. The performance studies are concentrated in 4 points: (1) Moving Object's Localization Efficiency; (2) Aggregation computing efficiency; (3) View trees' creating performance; (4) Searching fastest path performance.

Due to the limited paper space, we give only the comparison results respectively in figure6-9. These figures show evidently that HAT performs better than the other two indexing methods not only in the aspect of structure size, but also in querying processing.

Table 1. Parameters of the experiments

Parameter	Setting	Meaning
Page size	4K	The size of disk page and leaf node
Leaf capacity	90	Maximum number of road segments one leaf node(HAT, PMR Quadtree) or one block (Grid) could contains
X	[3420584, 4725764]	The range of coordinate x in indexed space domain
Y	[4077155, 5442669]	The range of coordinate y in indexed space domain
N	3000,...12000,..., 21000, 24123	The number of road segments in the map
M	50k,... 250k ...500k	The amount of generated moving objects
Referenced area size	1%,...9%... 81%	The proportion of referenced area related to total space

6 Conclusion

This paper has proposed a novel indexing technique HAT, in order to improve the efficiency of dynamic navigation techniques. In essence, it is a balanced Quadtree that adapts to the road network distribution, attached with pre-aggregated traffic measurement by its spatial hierarchy.

This feature allows the navigation process to filter the areas having high density traffic, without drilling down to check detail traffic information of each road segment. Thus, it can efficiently perform path searching process from macro to micro. Another contribution of this paper is the concept of “view tree” that restricts the optimal finding computation only within a part of the data structure that interests the user. Based upon the above techniques, we have proposed a system architecture, *DyNSA*, allowing dynamic navigation service in the prospect of Olympic Games in Beijing. The implementation and the experimentation results have validated our approach and demonstrated its efficiency and effectiveness. In perspective, we will test the HAT index in the context of data warehouses, and particularly to optimize OLAP like operator similarly to *aR-tree* proposed by Papadias *et al.* [9]. It would be also interesting to explore the possibility to combine real-time data on the traffic state and historical data to allow prediction of traffic state and adapt the navigation accordingly.

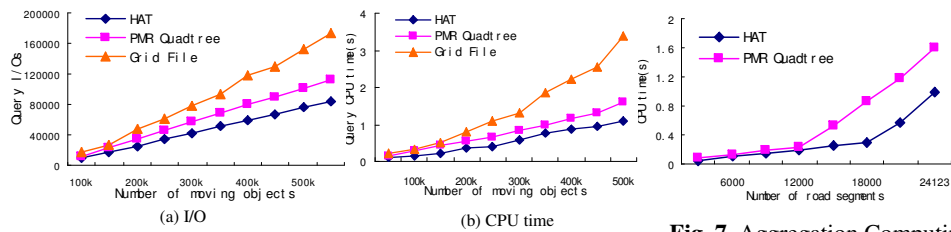


Fig. 6. Efficiency of Query Performance

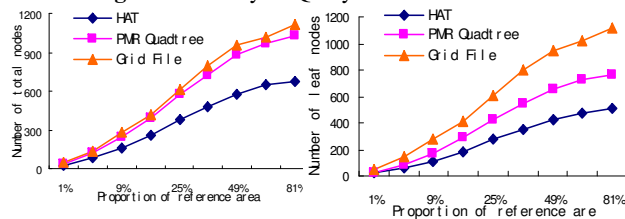


Fig. 8. Creating the view tree performance

Fig. 7. Aggregation Computing

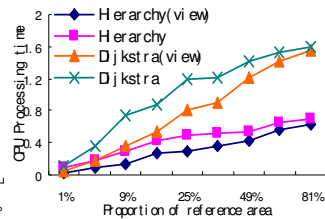


Fig. 9. Optimal Path Finding

Reference

1. Barbara D.: Mobile Computing and Databases - A Survey. IEEE Transactions on Knowledge and Data Engineering, (1999), Jan/Feb, Vol. 11(1) 108-117.
2. Chon, H., Agrawal, D., and Abbadi, A. E.: FATES: Finding A Time dEpendent Shortest path. Proc. of Int. Conf. on Mobile Data Management, (2003), 165-180.
3. Deo, N. and Pang, C.Y.: Shortest path algorithms: taxonomy and annotation, Networks, (1984) vol. 14, 275-323.
4. Fu, L. and Rilett, L.R.: Expected shortest paths in dynamic and stochastic traffic networks. Transportation Research, Part B: Methodological, (1996), vol. 32(7), 499-516.
5. Dijkstra, E. W.: A note on two problems in Connection with graphs, Numerische Mathematik, (1959), vol. (1), 269-271.
6. Tayeb, J., Ulusoy, O., Wolfson, O.: A Quadtree-based Dynamic Attribute Indexing Method. The Computer Journal, (1998), 185-200.
7. Nievergelt, J., Hinterberger, H.: The Grid File: An Adaptable, Symmetric Multikey File Structure. Proc. of ACM Trans. On Database Systems, Vol9(1), (984), 38-71.
8. Brinkhoff, T.: Network-based Generator of Moving Objects. <http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator>
9. Papadias D., Kalnis P., Tao Y., Efficient OLAP Operations in Spatial Data Warehouses. In Symposium on Spatial and Temporal Databases, 2001.