Graph Strip Tree for Efficient Search of Objects Moving on a Graph

by

Thuy T. T. Le and Bradford G. Nickerson

TR10_199, June 28, 2010

Faculty of Computer Science

University of New Brunswick

Fredericton, N.B. E3B 5A3

Canada

Phone: (506) 453-4566

Fax: (506) 453-3566

Email: fcs@unb.ca

www: http://www.cs.unb.ca

Graph Strip Tree for Efficient Search of Objects Moving on a Graph

Thuy Thi Thu Le and Bradford G. Nickerson

{m6839,bgn}@unb.ca
Faculty of Computer Science, University of New Brunswick
P.O. Box 4400, Fredericton, N.B. Canada E3B 5A3.

Abstract. A spatio-temporal data structure to index objects moving at constant velocity on a graph is presented. It is designed to efficiently answer rectangle R plus time instance and time interval queries about the past positions of moving objects. Such data structures are useful, for example, when searching for vehicles moving on a road network in specific areas at specific times. Unlike other data structures that use R-trees to index bounding boxes of moving object trajectories, our data structure represents moving objects as lines in a bounded space. For n moving object instances (unique entries of moving objects) on a graph with $|E|$ edges, we show that $O(\log_2 |E| + |L| \cdot \log_2(n/|E|) + k)$ time is required to answer a rectangle R plus time interval query, for $|L|$ the number of edges intersected by R and k the number of moving objects in range. Space $O(n + \lambda + |E|)$ is required to store n moving object instances with λ intersections among them in $|E|$ ordered polyline trees. Space $\Omega(n + |E|)$ is required to store the history of all n moving object instances.

1 Introduction

Research on efficient storage and retrieval of moving objects has many practical applications, such as tracking people in videos for security reasons, observing moving clouds for weather forecasting, and searching for moving vehicles on a road network for traffic planning, monitoring and simulation. Indexing a large number of moving objects to improve the response time for query processing becomes a significant challenge. Indexing can be done on either current and future positions of moving objects, or historical positions of moving objects [16]. Our research addresses the latter category. Queries on historical data are likely to be used in applications such as planning, event reconstruction and training. Much previous work on indexing moving objects assumes free movement of the objects in space (e.g., [2], [12], [14], [17], [18]). If movement is restricted to edges of a graph, the index should be able to use less storage than would be required if objects were free to move anywhere in space. We address the problem of indexing moving objects on a graph defined by its edges and vertices. The graph can be non-planar as it is when representing road networks [4].

Existing work on indexing objects moving on a graph include the MON-tree [3], PPFI [5], FNR-tree [6], and [13]. The common point of these data structures

id	**$[t_1,t_2]$**	**$[r_1,r_2]$**
o_1	5 10	0.5 0.8
o_2	0 9	0.7 1
o_3	10 15	0.6 0.4
o_4	5 13	0.3 0
o_5	16 25	1 0.6
o_6	2 22	0 1
o_7	12 25	0 0.68
o_8	3 23	0 1
o_9	5 25	0 1
o_{10}	1 21	0 1
o_{11}	5 25	1 0
o_{12}	0 20	1 0
o_{13}	4 24	1 0
o_{14}	3 23	1 0

(a)

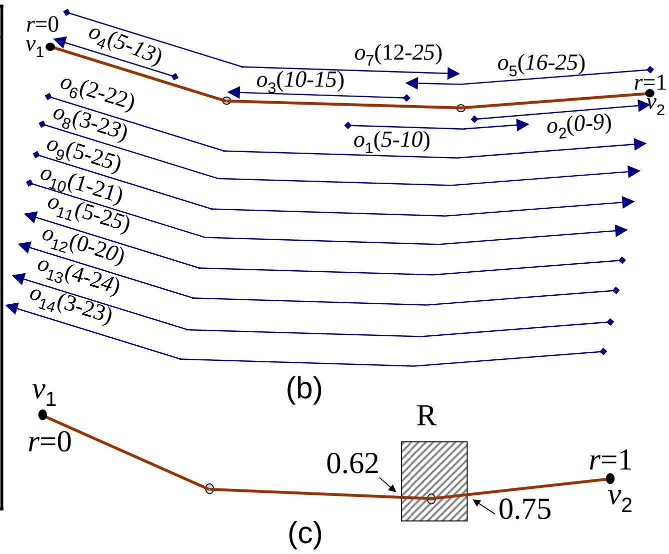

Fig. 1. (a) Table of 14 moving objects from (b) on edge connecting two vertices v_1 and v_2. Each directed polyline represents a position interval of an object with a direction. The two numbers in parentheses represent time intervals of corresponding objects. (c) rectangle query R, where R intersects the edge at the position interval $[r_1, r_2]$=[0.62, 0.75] and the time interval query is $[t_1, t_2]$=[23, 25].

is to combine several R-trees to index moving objects on a fixed network. A network is indexed by an R-tree [15] [9] while moving objects are indexed on a forest of R-trees, whose roots are linked to leaf nodes of the network tree. A moving object is represented as a (space × time) rectangle whose one side is a time interval and whose other side is a position interval of that moving object. The disadvantage of these data structures is that the number of retrieved objects for a query can be much more than the exact result. When a moving object rectangle intersects a (space × time) query rectangle, we still do not know for certain whether this moving object is in range or not. The time complexity of this approach is controlled by the number of (space × time) moving object rectangles intersecting the (space × time) query. Fig. 1 show an example of 14 moving objects on an edge, and Fig. 1 (c) shows a query rectangle R=([23,25],[0.62,0.75]), respectively. There are 6 moving objects having their begin or end positions falling inside the edge. Eight moving objects $o_6, o_8, ..., o_{14}$ move across the entire edge. A diagonal line segment also represents the direction of moving objects at a constant velocity. Diagonal line segments from upper left to lower right represent objects moving from the right ($r = 1$) to left ($r = 0$) direction. When rectangles are used to index these moving objects, five rectangles representing five moving objects o_5, o_7, o_9, o_{11} and o_{13} intersect with the shaded query rectangle R as shown in Fig. 2. However, only two moving objects o_5 and o_7, whose rectangle's diagonal line segments intersect with R, are actually in range. Nine other moving

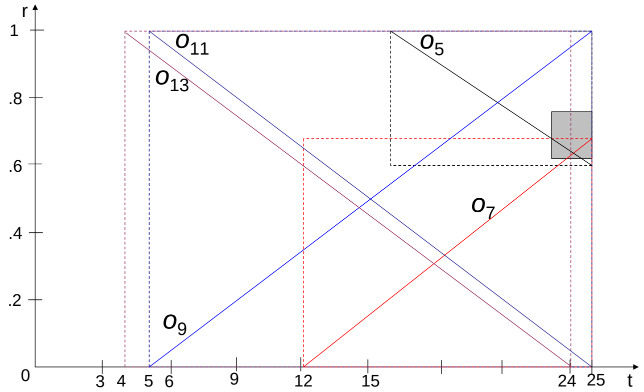

Fig. 2. Five $[t_1^j, t_2^j]$, $[r_1^j, r_2^j]$ rectangles represent five moving objects o_5, o_7, o_9, o_{11}, and o_{13} from Fig. 1(a). The solid line segments illustrate diagonal line segments of rectangles (shown by dashed lines). The shaded rectangle is a query rectangle.

objects o_1, .., o_4, o_6, o_8, o_{10}, o_{12}, and o_{14} are not shown in Fig. 2 because their rectangles do not intersect with R. In the worst case, all moving object rectangles on an edge intersect the query rectangle R, but none of them is in range.

We propose a new data structure that allows us to exactly retrieve moving objects for a query. Instead of using R-trees to index bounding boxes of moving objects, we index oriented and bounded lines representing positions of moving objects at different times. With this new data structure, we can answer a rectangle R plus time interval query in $O(\log_2 |E| + |L| \log_2(n/|L|) + k)$ time, where n is the number of moving object instances (unique entries of moving objects) on a graph with $|E|$ edges, $|L|$ is the number of edges intersected by R and k is the number of lines containing moving object instances in range. This data structure improves the search time complexity of our previous result [7].

None of the previous research reports worst case query time, but they all depend on R-tree indexing for spatial search of the graph which requires $\Omega(|E|^{\frac{1}{2}})$ time. Moreover, objects moving on edges of the graph are indexed using 2-d R-trees which requires $O(n^{\frac{1}{2}} + k^{'})$ time for searching in-range objects in each R-tree, for $k^{'}$ is the number of rectangles representing moving objects intersecting the (time interval $\times$ position interval) query on an edge. If $|L|$ edges of the graph intersect the spatial query, the total worst case time to search all rectangles representing moving objects intersecting the query on $|L|$ edges is $O(|E|^{\frac{1}{2}} + (n_1^{\frac{1}{2}} + k_1^{'}) + (n_2^{\frac{1}{2}} + k_2^{'}) + .. + (n_{|L|}^{\frac{1}{2}} + k_{|L|}^{'}))$ $=O(|E|^{\frac{1}{2}} + (n_1^{\frac{1}{2}} + n_2^{\frac{1}{2}} + .. + n_{|L|}^{\frac{1}{2}}) + (k_1^{'} + k_2^{'} + .. + k_{|L|}^{'}))$ $=O(|E|^{\frac{1}{2}} + |L| n^{\frac{1}{2}} + k^{'})$. Here, $k^{'} = \sum_{i=1}^{|L|} k_i^{'}$ is the total number of rectangles representing moving object instances intersecting the query. This algorithm does not apply to the FNR-tree which uses B-trees (1D R-trees) for temporal search. In the worst case, the FNR-tree requires $O(|E|^{\frac{1}{2}} + (\log_2 n_1 + \log_2 n_2 + .. + \log_2 n_{|L|}) + (k_1^{'} + k_2^{'} + .. + k_{|L|}^{'}))$$=O(|E|^{\frac{1}{2}} + |L| \log_2(n/|L|) + k^{'})$ search time. Note that $k^{'}$ is significantly larger than k as previous techniques can falsely report intersections with R (see Fig. 2).

2 Proposed Approach

We support two types of queries: time instant queries defined as $Q_1 = (R, t_q)$ to find the k moving objects intersecting rectangle R at time t_q, and time interval queries defined as $Q_2 = (R, [t_1, t_2])$ to find the k moving objects intersecting rectangle R at any time during time interval $[t_1, t_2]$. Both query types can be counting queries (report only k) or reporting queries (report the identity of the k moving objects satisfying the query). A spatio-temporal query $Q_2 = (R, [t_1, t_2])$ is transformed to a query rectangles $Q_3 = [t_1, t_2] \times [r_1, r_2]$ by finding positions r_1, r_2 that span the query rectangle R on an edge. In Fig. 3 (a) and (b), a query Q_2 on an edge is transformed to two query rectangles Q_3.

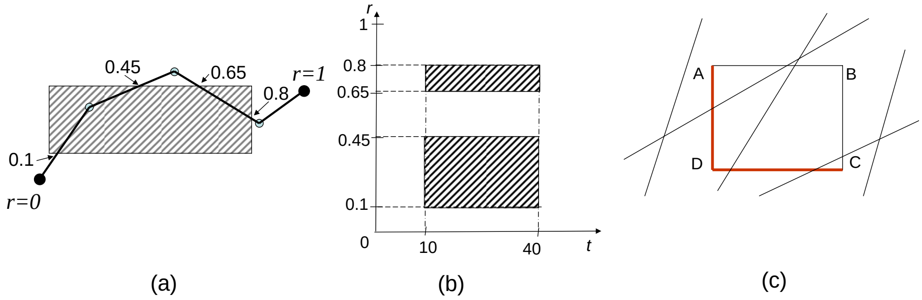

Fig. 3. (a) A spatio-temporal query $Q_2 = (R, [10, 40])$ is transformed into (b) two query rectangles Q_3: [10,40]×[0.1, 0.45] and [10,40]×[0.65, 0.8]. (c) Query rectangle Q_3 with four vertices A, B, C, and D. Lines intersect the rectangle Q_3 if and only if they intersect line segments AD or DC of the rectangle.

Assume that objects move at a constant velocity on an edge. Each moving object is represented by a diagonal line segment of the (time interval) × (position interval) rectangle. Each point on this line segment corresponds to a position of a moving object at a specific time. If a line segment intersects a query rectangle, its corresponding moving object is in range. For example, because the two diagonal line segments of two rectangles of o_5 and o_7 intersect the shaded rectangle query (Fig. 2), only objects o_5 and o_7 are in range. A challenge is how to index line segments efficiently to achieve an efficient search on moving objects. A recent work [10] indexes line segments by two B^+-trees, one to store $x-$coordinates and the other to store $y-$coordinates of end points of all line segments. However, a rectangular search on two B^+-trees independently may result in inefficient search time if all nodes in one tree are in range while none of the nodes in the other tree is in range. In other words, all nodes of one B^+-tree are visited even though none of them is in range.

We present a new method to index bounded lines representing moving objects on an edge. When objects move across an edge from $r = 0$ to $r = 1$, their corresponding path on that edge is considered as a line in a bounded plane formed by $(time \times r)$, where $0 \leq time \leq T$, $0 \leq r \leq 1$. For simplicity, we use the terminology bounded lines or lines in this paper to imply lines representing moving objects in the bounded plane.

Assume that we have a set of lines having slopes $m \in (0, \infty]$ in a bounded plane as in Fig. 3(c). If we want to find lines that intersect a query rectangle Q_3 with four vertices A, B, C, and D, we only need to find lines intersecting line segments AD and DC of Q_3. From this idea, we divide the set of bounded lines into two subsets L_1 and L_2. Each subset contains lines of objects moving in the same direction on the edge. L_1 is the subset moving from $r = 0$ to $r = 1$, and L_2 is the subset moving from $r = 1$ to $r = 0$. In the following discussion of the paper, we will focus only on L_1. This assumption provides the basis for our data structure. For objects in L_2 (e.g., o_5, o_{11}, o_{13} in Fig. 2), the ordered polylines would divide the (t, r) space in a monotonic decreasing fashion. Algorithms and analysis for L_1 are similarly applied to L_2.

We use the notion t-level(i) to refer to set of lines intersecting line $x = i$ ordered top-to-bottom. Similarly, r-level(i) refers to a set of lines intersecting line $y = i$ ordered left-to-right. Fig. 4 shows an example of two t-levels: t-level(9.2) and t-level(10.4), and two r-levels: r-level(0.35) and r-level(0.62). The order of lines is potentially different for different values of i.

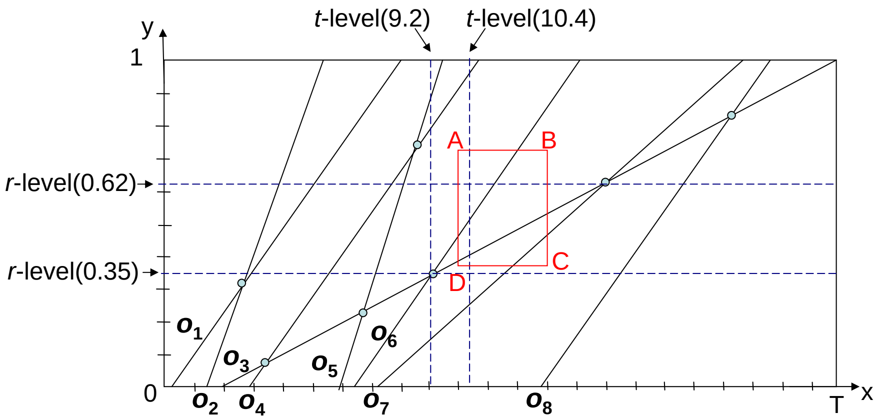

Fig. 4. Example of a query rectangle Q_3, with points A=(10,0.72), B=(13,0.72), C=(13,0.37), and D=(10,0.37), on a set 8 bounded lines. Dashed lines shows t-levels and r-levels near two line segments AD and DC of Q_3. Since lines o_3 and o_6 intersect AD, and line o_7 intersects DC, these three lines are in range.

Consider a set of lines and a query rectangle Q_3 in Fig. 4, we only need to search for lines intersecting AD on t-level(9.2) and DC on r-level(0.35). We build a data structure for efficient search based on this idea.

Given a set of intersection points, we present a method to organize them efficiently. When lines intersect each other, they form ordered polylines. An ordered polyline p_i is created by connecting parts of lines at intersections (with each other and with the $r = 0$, and $r = 1$ boundaries). For example, the first four ordered polylines in Fig. 5 are $p_1 = \{o_{1.1}, o_{2.2}\}$, $p_2 = \{o_{2.1}, o_{1.2}\}$, $p_3 =$

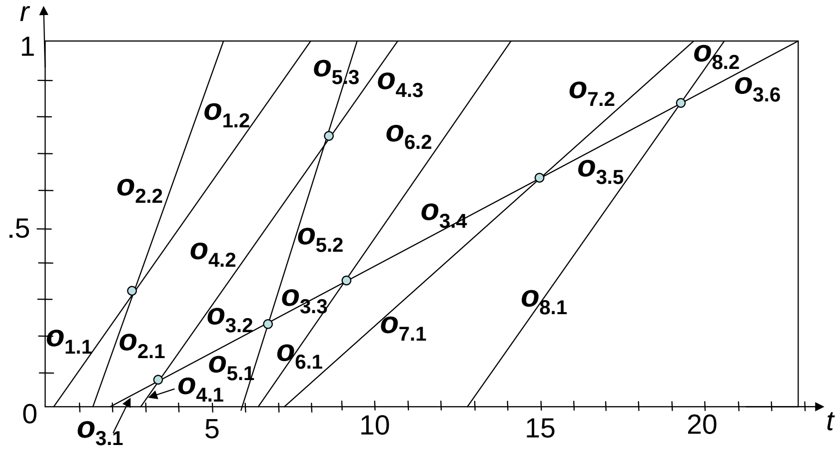

Fig. 5. Lines represent 8 moving objects traveling at a constant velocity over an edge in direction from $r = 0$ to $r = 1$. $o_{j.i}$ belongs to the line representing moving object o_j.

$\{o_{3.1}, o_{4.2}, o_{5.3}\}$, and $p_4 = \{o_{4.1}, o_{3.2}, o_{5.2}, o_{4.3}\}$, ordered from left to right; they do not intersect each other. Points in an ordered polyline are monotonically increasing in both t and r. We connect points in an ordered polyline together into a list of entries, and arrange ordered polylines in a balanced search tree. An entry of an ordered polyline points to entries, having the same or near values, of its left, right and next polylines. Section 3 shows the details of the data structure.

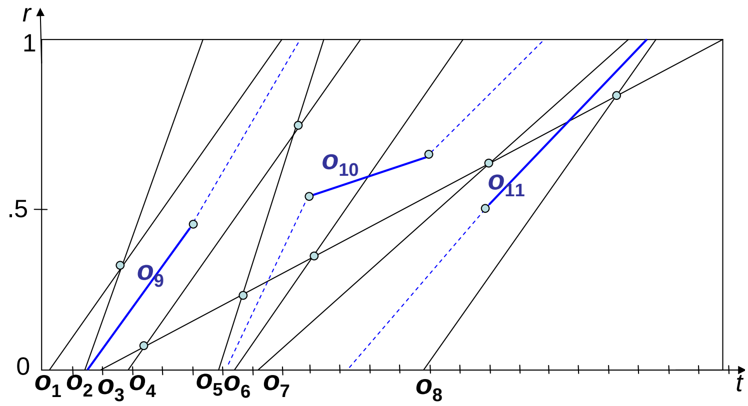

Fig. 6. Lines represent all situations of moving objects. Objects o_9, o_{10}, and o_{11} are extended, and induce new ordered polylines that account for intersections with ordered polylines spanning the entire edge (i.e. with $r \in [0, 1]$).

When we consider historical positions of vehicles on a road network, most vehicles move from the start to the end of the edge representing a road. Others may start to move or stop in the middle of the road. We consider all positions of moving objects in our data structures. Ordered polylines also work for objects

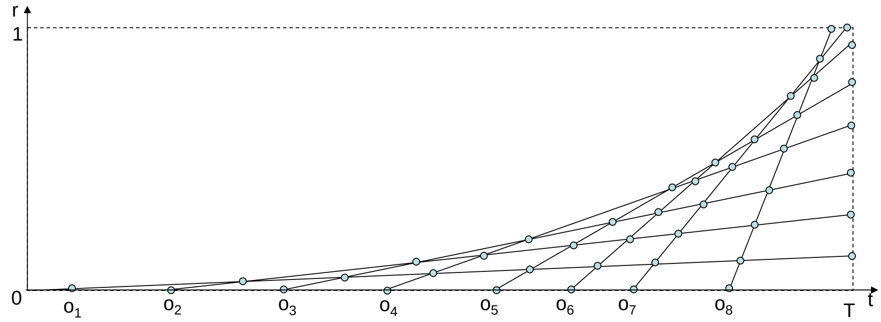

Fig. 7. Example of 8 polylines representing 8 moving object instances o_1, ..., o_8 in the worst case, where each object instance intersects 7 others in time.

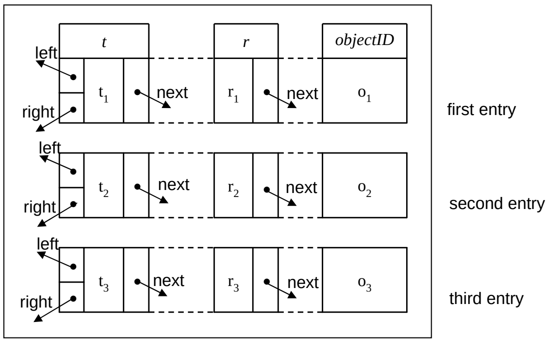

Fig. 8. Structure of a node containing three entries in an ordered polyline tree.

moving on a road, stopping for a period of time, then moving again. For objects not moving over the entire edge (e.g., o_9, o_{10}, and o_{11} in Fig. 6), we add one (or two) line segment(s) to their line segments' endpoint(s) so that the connected parts from these original line segments reach the $r = 0$ and $r = 1$ boundaries. The extending line segment starts from one endpoint, whose r-coordinate is not 0 or 1, to a point having r-coordinate=0 or r-coordinate=1, and t-coordinate falling half-way between the end points of the two adjacent bounded lines. Fig. 6 shows an example of three extended line segments for objects o_9, o_{10}, and o_{11}.

In the worst case, every line representing a moving object instance intersects the lines representing all other moving object instances on the same edge (see Fig. 7). There are $O(g_i^2)$ lines for g_i moving object instances on edge i. Each ordered polyline requires $O(g_i)$ lines. The number of ordered polylines is still precisely g_i.

3 The Primary Data Structure

3.1 Indexing moving objects on an edge

Ordered polylines are arranged as a balanced binary search tree, called *ordered polyline tree*, based on each p_i dividing the space. Each ordered polyline con-

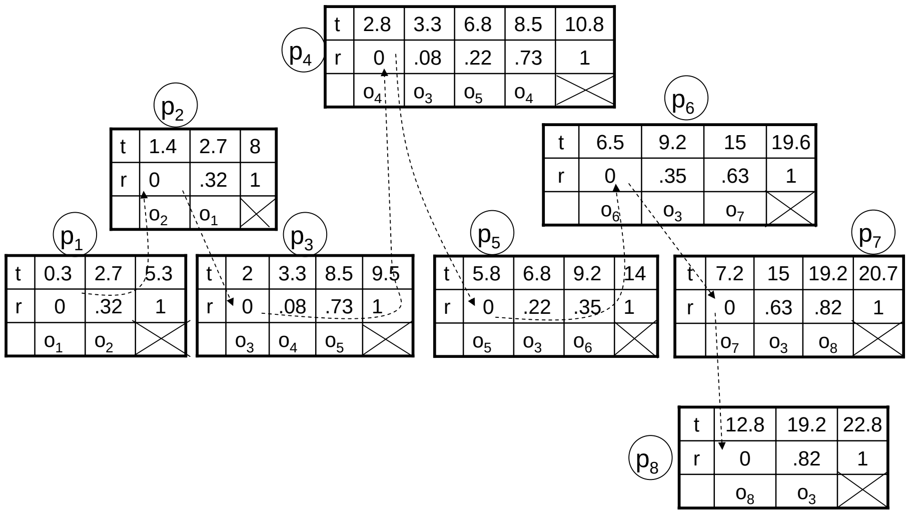

Fig. 9. Ordered polyline tree indexes 8 bounded lines in Fig. 5 at r-level(0). A three-row rectangle represents an ordered polyline, where each column is a point represented as an entry. A dashed line represents a pointer of an entry to its next adjacent entry. The cross symbol in the lower-right corner of each rectangle means that there is no $objectID$ in the last entry of each ordered polyline.

tains a list of entries. Each entry contains a point (t, r), a moving object ID corresponding to the line connecting to the point, three t-pointers, and one r-pointer (Fig. 8). Three left, right, and next t-pointers point to the left, right, and next adjacent entries (belong to the left, right, next adjacent ordered polylines), respectively on t-levels. One next r-pointer points to the next adjacent entry belonging to the next adjacent ordered polyline on r-levels.

For a polyline p_i with t-entry t_j, the (left, right, next) pointers point to the largest t-entry in p_i's (left, right, next) node $\leq t_j$, respectively. If no t-entries in p_i's (left, right, next) nodes are $\leq t_j$, the (left, right, next) pointers point to the smallest t-entry $> t_j$. In this way, we record all line segments in the arrangement of bounded lines such that a traversal of the tree from root to leaf serves to find the polyline immediately to the left of a query point A. Following next pointers of t-entries finds segments of ordered polylines in downward order for a vertical query segment AD. Following next pointers of y-entries finds segments of ordered polylines in left-to-right order for a horizontal query segment DC. Fig. 9 show an example of an ordered polyline tree on t-level(0). Ordered polyline trees can be made dynamic as presented in [8].

3.2 Indexing edges of a graph

An edge on a fixed graph $G = (V, E)$ is considered as a polyline. We index each edge by a strip tree [1]. Strip trees created are merged bottom up in pairs to construct a *graph strip tree*. Fig. 11 shows an example of strip trees created from a fixed graph in Fig. 10, and Fig. 12 a graph strip tree from the strip trees

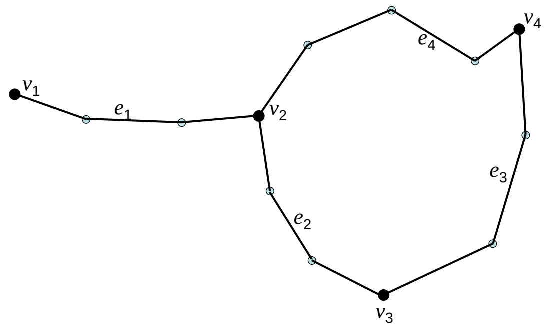

Fig. 10. Example graph G with 4 edges $e_1, ..., e_4$ and 4 vertices $v_1, ..., v_4$.

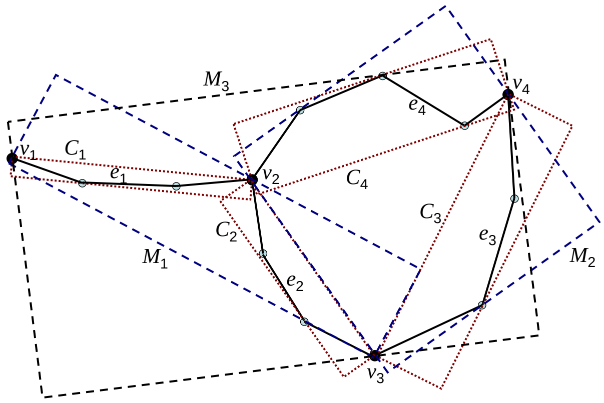

Fig. 11. Edges of graph G (Fig. 10) are represented as strip trees, with $C_1, ..., C_4$ representing the root bounding boxes for each strip tree. The strip trees are merged bottom up in pairs to construct a graph strip tree.

merged. Leaf nodes C_i point to strip tree S_i spatially indexing e_i and to ordered polyline tree T_i indexing lines representing moving objects on e_i.

4 Space Complexity

4.1 Storage space for an ordered polyline tree

Assume there is a set of g_i lines representing g_i objects moving in the direction from $r = 0$ to $r = 1$ in edge e_i with λ_i intersections among them.

Theorem 1 *An ordered polyline tree uses $O(g_i + \lambda_i)$ space to index a set of g_i lines with λ_i intersections among the lines.*

Proof. The number of entries in an ordered polyline is the number of intersection points forming the ordered polyline, plus two end points of the ordered polyline. There are g_i ordered polylines formed from the intersection of the g_i lines. Each intersection point (between two lines) belongs to two ordered polylines. If λ_i is the number of intersections among g_i lines, the number of entries in all the ordered polylines is $2g_i + 2\lambda_i$, or $2(g_i + \lambda_i)$. Assume each element of an entry of a node has size 1 (e.g., 1 for an coordinate t or r, a object ID, or a pointer). The size of an entry of an ordered polyline tree is 7. Therefore, the size of the tree is $7 \times 2(g_i + \lambda_i) = 14(g_i + \lambda_i)$, or $O(g_i + \lambda_i)$.□

Note that if c lines intersect at a point, where $c > 2$ and c is a constant number, we insert c entries to the tree for all c even; we insert $c - 1$ entries to the tree for all c odd because the middle line does not change its order at the intersection point. If c is the maximum number of lines involved in an intersection, at most $c\lambda$ entries are inserted for λ intersections because at most c entries are inserted at one intersection. The space required for the tree is $O(7(2g_i + c\lambda_i)) = O(n + \lambda)$. Theorem 1 still holds. In a special case when $c = g_i$ and $\lambda_i = 1$ (i.e., g_i lines intersect at one point), those g_i lines change their order after the intersection point. Therefore g_i (or $g_i - 1$) new entries are inserted to the tree at once. The total entries of the tree in this case is $O(2g_i + g_i)$=$O(g_i)$. Theorem 1 still holds with $\lambda_i = 1$.

We see that the space for an ordered polyline tree in the best case is $O(g_i)$ when there are at most αg_i intersections (i.e.,$O(g_i + \alpha g_i)$=$O(g_i)$, for $\alpha \geq 0$ a constant). In the worst case where the number of intersections is λ=$O(g_i^2)$, the space required is $O(g_i + g_i^2)$=$O(g_i^2)$.

4.2 Storage space for the entire graph strip tree

We store the $|E|$ edges of a graph in a graph strip tree (see Figures 10, 11 and 12) requiring $O(1)$ space for each edge. This is a reasonable assumption based on actual road network statistics. For example, the number of edges in the entire road network of Canada [19] is $|E|$ = 1,869,898, with an average of 7.32 segments per edge. If we assume that a merged strip tree is built from $|E|$ = 2,000,000, and each strip tree requires 1,000 bytes (a generous allocation), a main memory size of 2 GB will suffice to hold the merged strip tree.

Theorem 2 *A graph strip tree with $|E|$ edges uses $O(|E|+n+\lambda)$ space to index a set of n moving object instances with λ intersections among lines representing moving objects.*

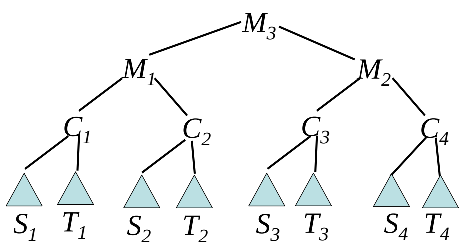

Fig. 12. The graph strip tree corresponding to the graph in Fig. 11. Each leaf C_i points to strip tree S_i representing edge e_i, and ordered polyline tree T_i indexing moving objects on e_i.

Proof. $|E|$ edges of the graph indexed by a graph strip tree require $O(|E|)$ space. Each ordered polyline tree with g_i moving objects and λ_i intersections requires $O(g_i+\lambda_i)$ space (Theorem 1). The graph strip tree containing $|E|$ ordered polyline trees requires $O(\Sigma_{i=1}^{|E|}(g_i+\lambda_i)) = O(\Sigma_{i=1}^{|E|}(g_i)+\Sigma_{i=1}^{|E|}(\lambda_i))$=$O(n+\lambda)$ space. Therefore, the graph strip tree requires $O(|E|+n+\lambda)$ space. □

5 Search Complexity

5.1 Searching on an ordered polyline tree

Given query rectangle Q_3 with four vertices A, B, C, D with $D=(x,y)$ in the clockwise direction, searching finds lines in t-level(x) intersecting AD and lines in r-level(y) intersecting DC. The main steps of the search algorithm are as follows:

(1) Starting from the root node, choose the entry b with largest t-value $\leq x$. If $x <$ smallest t-value, choose the smallest entry.
(2) Follow the b's $left$ or $right$ pointer to the next entry b' by comparing line *objectID* of entry b to point A. If A is left of the line, follow the left pointer; otherwise follow the right pointer.
(3) Compare the t-value of entry b' to t-values of entries following b'. Choose the largest entry b whose t-value $\leq x$.
(4) Repeat (2) and (3) until the entry b at a leaf node is reached. If b's line *objectID* intersects AD, report it.
(5) Use the *next* t-pointer at b to go to the entry b' on the adjacent polyline. Compare the t-value of entry b' to t-values of its neighboring entries a' and c'. Choose the entry b among a', b', c' having the maximum t-value $\leq x$. If b's line *objectID* intersects AD, report it.
(6) Repeat (5) until an entry having its line not intersecting AD is reached.
(7) At the current node, choose the entry b having a maximum r-value $\leq y$.
(8) Report *objectID* of the entry b if its line intersects DC.
(9) Use the *next* r-pointer at b to go to the next adjacent entry, again called b'. Compare the r-value of entry b' to r-values of its neighboring entries a' and c'. Choose the entry b among a', b', c' having the maximum r-value $\leq y$. If b's line *objectID* intersects AD, report it.
(10) Repeat (9) until an entry having its line not intersecting DC is reached.

The algorithm stops if a right-most leaf node is reached at any step. Consider performing the query rectangle Q_3 in Fig. 4 on a set of 8 bounded lines. The algorithm needs to search for lines at entries containing t-levels(10) and r-level(0.37) intersecting AD and DC, respectively. We start from entry b=(8.5,0.72, o_4) of the root node p_4 (Fig. 9). We do not report o_4 in range because it does not intersect AD. As A is on the right side of o_4, we follow the right t-pointer of b to entry b'=(5.5, 0, o_6) of node p_6. Since entry c'=(9.2,0.35, o_3) next to b' has maximum t-value $\leq$ 10, we choose entry c' to query. We report o_3 because it intersects AD. As A is on the left side of o_3, we follow the left t-pointer of o_3 to a new entry b'=(9.2, 0.35, o_6) of node p_5. We report o_6 because it intersects AD. Because p_5 is a leaf node, we follow the next t-pointer of o_6 to entry b'=(9.2, 0.35, o_3) of node p_6. We skip p_6 because its entry o_3 is already marked as a reported node. We follow the next t-pointer of o_3 to entry b'=(7.2, 0, o_7) of node p_7. Since o_7 does not intersect AD, we start to search for lines intersecting DC at r-level(0.37). When comparing r-value=0 of b' and r-value=0.63 of entry c'=(12, 0.63,o_8), 0 is the maximum r-value ≤ 0.37, we still choose b' to query. We report o_7 because it intersects DC. Finally, we follow the next r-pointer of o_7 to entry b'=(12.8,0, o_8) of node p_8. We stop the search since o_8 does not intersects DC. There are a total of three moving objects o_3, o_6, and o_7 in range.

Theorem 3 *For a single edge, and assuming objects move at a constant velocity, the time to report moving objects intersecting a query rectangle Q_3 on the ordered polyline tree T_i is $O(\log_2(g_i)+k_i)$, where g_i is the number of moving objects stored in T_i, and k_i is the number of moving objects in range.*

Proof. Let w be the number entries forming an ordered polyline. We consider all objects moving on an edge to be unique. If the same moving object crosses the same edge at different time intervals $[t_1, t_2]$ and $[t_3, t_4]$, we consider them as different moving objects. These duplicated moving objects are defined as moving object instances in Theorem 4. Consider the 10 steps of the searching algorithm above. Step (1) requires $O(log_2 w)$ time. Steps (2), (3) and (4) take $O(\log_2 g_i)$ time to reach a leaf. Steps (5), (7), (8) and (9) take $O(1)$ time, and report the k lines intersecting Q_3. If k' lines intersect AD, steps (5) and (6) require $O(k')$ time to report them. If k'' lines intersect DC, steps (9) and (10) require $O(k'')$ time to report them. Therefore, the total required time for searching is $O(log_2(w)+\log_2 g_i+k'+k'')$= $O(log_2(g_i)+k_i)$ since $w \leq g_i$, and $k'+k'' \leq k_i$.□

5.2 Searching on a graph strip tree

We assume that there are n object instances moving on $|E|$ edges of a graph over a time domain $[0, T]$. Combining these assumptions with Theorem 2 leads to the following theorem:

Theorem 4 *There exists a data structure indexing objects moving on a graph that answers a Q_2 query in time $O(\log_2 |E| + |L| \log_2(\frac{n}{|L|}) + k)$, for k the number of moving object instances in range, and $|L|$ the number of edges on the graph intersecting Q_2.*

Proof. Searching for moving objects intersecting a query rectangle Q_2 and a time interval $[t_1, t_2]$ starts from the root of the strip tree, and returns a list L of edges intersecting Q_2. This searching requires $O(\log_2 |E|)$ time since we assume a constant number of segments defining an edge, thus, a single strip tree needs $O(1)$ storage space and $O(1)$ time for searching.

As we assume a constant number of segments define an edge, a query Q_2 on an edge is transformed to a constant number of Q_3 queries on the same edge. Performing a Q_2 query or a constant number of queries Q_3 on each of the $|L|$

ordered polyline trees requires $O(\log_2(g_i) + k_i)$ time (Theorem 3), where k_i is the number of lines representing moving object instances in range in T_i.

The search time of $|L|$ edges in the T_i trees is $O(\sum_{i=1}^{|L|}(log_2(g_i)+k_i) = O(\log_2(g_1)+\log_2(g_2)+...+\log_2(g_{|L|}))+k) = O(\log_2(g_1\times g_2\times...\times g_{|L|})+k)$, where $k=\sum_{i=1}^{L}(k_i)$ is the total number of moving object instances in range.

According to the AM-GM inequality rule [11], $\sqrt[|L|]{g_1 \times g_2 \times ... \times g_{|L|}} \leq \frac{g_1+g_2+...+g_{|L|}}{|L|} \leq \frac{g_1+g_2+...+g_{|E|}}{|L|} = \frac{n}{|L|}$, where $|L| \leq |E|$, or $(g_1\times g_2\times...\times g_{|L|}) \leq (\frac{n}{|L|})^{|L|}$. Therefore, $\log_2(g_1\times g_2\times...\times g_{|L|}) \leq \log_2((\frac{n}{|L|})^{|L|}) = |L|\log_2(\frac{n}{|L|})$. Thus, the search time of $|L|$ edges is $O(|L|\log_2(\frac{n}{|L|}) + k)$. Combining with the time to search the strip trees $O(\log_2|E| + |L|)$, the time to search the graph strip tree is $O(\log_2|E| + |L| + |L|\log_2(\frac{n}{|L|}) + k)$, or $O(\log_2|E| + |L|\log_2(\frac{n}{|L|}) + k)$. $\square$

If the number of moving object instances is much greater than the number of edges on the graph (i.e., $n \gg |E|$), we expect the search time to be dominated by the time $O(|L|\log_2(\frac{n}{|L|}) + k)$ to search $|L|$ ordered polyline trees. Note that in the worst case, n moving object instances fall on a single graph edge e_i, and the spatial query intersects e_i. If these n moving objects happen to be uniformly distributed among the $|E|$ edges, the query time becomes $O(\log_2|E| + |L|\log_2(\frac{n}{|E|}) + k)$.

6 Conclusion

We present a new data structure for efficient search of objects moving on a graph. The underlying graph can be non-planar. Our data structure is a combination of strip trees and ordered polyline trees. Strip trees are used for spatial indexing of the graph edges. Each strip tree at leaf level represents a polyline corresponding to a road or road segment in a road network. Ordered polyline trees are used to index the trajectories of moving objects on one of the graph edges.

Unlike previous data structures using rectangles to represent moving objects, we use bounded lines. The main advantage of our data structure is that it can answer a rectangle R plus time interval $[t_1, t_2]$ query in an output sensitive fashion in expected time logarithmic in n. There are some other advantages for our data structure. First, the strip trees index the graph geometry, and the ordered polyline trees index bounded lines representing moving objects are independent. One can update the ordered polyline trees without changing the strip tree, or update a strip tree when an edge geometry changes, without affecting other edge strip trees. Second, since moving objects on a graph edge belong to a strip tree, we can easily answer queries which count moving objects on a single edge. For example, we can count how many vehicles move on a specific road at a specific time or during a specific time interval.

An open problem is how to efficiently index moving object instances to achieve an I/O-efficient worst case optimal search complexity.

7 Acknowledgements

This research was supported, in part, by the Natural Sciences and Engineering Research Council (NSERC) of Canada, the UNB Faculty of Computer Science, the Harrison McCain Foundation, MADALGO - Center for Massive Data Algorithmics (a Center of the Danish National Research Foundation) and the government of Vietnam.

References

1. D. H. Ballard. Strip trees: a hierarchical representation for curves. *Communications of ACM*, 24(5):310–321, 1981.
2. H. D. Chon, D. Agrawal, and A. E. Abbadi. Query processing for moving objects with space-time grid storage model. In *MDM '02: Proceedings of the Third International Conference on Mobile Data Management*, page 121, Washington, DC, USA, 2002. IEEE Computer Society.
3. V. T. de Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.
4. D. Eppstein and M. T. Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *ACM GIS '08*, pages 1–10, November, 5-7 2008.
5. Y. Fang, J. Cao, Y. Peng, and L. Wang. Indexing the past, present and future positions of moving objects on fixed networks. In *CSSE '08*, pages 524–527, Washington, DC, USA, 2008. IEEE Computer Society.
6. E. Frentzos. Indexing objects moving on fixed networks. In *Proceedings of the 8th International Symposium on Spatial and Temporal Databases*, pages 289–305, Santorini, Greece, July 24-27, 2003.
7. T. T. T. Le and B. G. Nickerson. Efficient Search of Moving Objects on a Planar Graph. In *ACM GIS '08*, pages 367–370, Irvine, CA, USA, November, 5-7 2008.
8. T. T. T. Le and B. G. Nickerson. Towards a Dynamic Data Structure for Efficient Bounded Line Range Search. In *Procedding of CCCG 2010*, Winnipeg, Manitoba, Canada, August 9-11 2010, in press.
9. S. Leutenegger and M. A. Lopez. *Chapter 12 - Handbook of Data Structures and Applications*. 2005.
10. H.-Y. Lin. Using b+-trees for processing of line segments in large spatial databases. *J. Intell. Inf. Syst.*, 31(1):35–52, 2008.
11. A. Lohwater. *Introduction to Inequalities*. 1982.
12. J. Ni and C. V. Ravishankar. Indexing spatio-temporal trajectories with efficient polynomial approximations. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):663–678, May 2007.
13. D. Pfoser and C. S. Jensen. Indexing of network constrained moving objects. In *Proceedings of the 11th ACM GIS*, pages 25–32, New Orleans, Louisiana, USA, November 07 - 08, 2003.
14. D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 395–406. Morgan Kaufmann, 2000.
15. P. Rigaux, M. Scholl, and A. Voisard. *Introduction to Spatial Databases: Applications to GIS*. Morgan Kaufmann, 2000.

16. S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference*, pages 331–342, Dallas, Texas, United States, May 15 - 18, 2000.
17. Z. Song and N. Roussopoulos. Hashing moving objects. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 161–172, London, UK, 2001. Springer-Verlag.
18. Z. Song and N. Roussopoulos. Seb-tree: An approach to index continuously moving objects. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 340–344, London, UK, 2003. Springer-Verlag.
19. Statistics Canada. 2009 road network file. www.statcan.gc.ca, last accessed: September 21, 2009.