A Data Structure for I/O Efficient Search of Objects Moving on a Graph

by

Thuy T. T. Le and Bradford G. Nickerson

TR09_192, April 30, 2009

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

Abstract

We present a spatio-temporal data structure called minimum I/O Graph Strip Tree (minGStree) to index moving objects on a graph. The minGStree is designed to efficiently answer time instance and time interval queries about the past positions of moving objects. The minGStree uses $\Theta(\frac{n}{B})$ blocks of external memory, where n is the number of moving object instances (unique entries of moving objects) and B is the I/O block size. For n moving object instances randomly distributed on the E edges of a graph over a time domain $[0, T]$, the expected number of I/Os required to determine the moving objects intersecting one edge (for both time instance queries and time interval queries) in a minGStree is $O(log_B(\frac{n}{E}) + k)$, where k is the number of disk blocks required to store the answer. We anticipate this data structure will be practical to implement.

1 Introduction

A significant challenge in spatio-temporal databases is how to improve the response time for query processing of moving objects. In these large databases, where data are stored in external memory, the response time depends on the number of I/Os (disk accesses) required to process queries. Saltenis et al [13] divide the problem of indexing the positions of continuously moving objects into two categories. Queries about the current and anticipated future positions of moving objects define one category. Such queries are likely to be used in real-time and near real-time systems. Applications such as traffic control, emergency response and navigation while driving fall into this category. The second category focusses on the history of the positions of moving objects. Queries on historical data are likely to be used in applications such as planning, event reconstruction and training. Our research addresses the latter category.

Two basic approaches are used to index moving objects. Indexing the trajectories of the objects, with updates of trajectories triggering index updates, permits storage and indexing of the paths of objects described as a function $p_i(t)$ of time t for moving point i. This is the approach followed by Agarwal et al [1], so that a feasibly sized index can be built for (potentially) many moving objects. The second approach considers updates arriving at regular intervals for all objects. Hadjieleftheriou et al [10] follow this approach. Regular interval updates simplify the update algorithm, but require more space to store object positions that may be a linear extrapolation of the two previous object positions.

Most previous work for indexing moving objects assumes free movement of the objects in space. 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. Our approach does this and updates moving objects twice on an edge (i.e., when objects enter or leave edges).

2 Our Results

We address the problem of indexing moving objects on a graph (possibly disconnected) defined by its edges and vertices. The graph can be non-planar as it is when representing road networks [8]. The minGStree data structure extends previous work that gave rise to the GStree [11]. The minGStree requires $\Theta(n/B)$ disk blocks by having only one time-based I/O efficient interval tree T_i per edge along with one position-based I/O efficient interval tree P_i per edge. 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).

For n moving object instances randomly distributed over E edges, we show that the I/Os required to determine the moving objects on an edge satisfying a query (Q_1 or Q_2) is expected to be $O(\log_B \frac{n}{E} + k)$, where k is the number of disk blocks required to store the answer.

3 Related Work

There has been significant research into indexes able to store complete histories of data repositories. Kinetic data structures [4] make search of complete histories of moving objects possible by updating the data structure only when significant kinetic events occur. For example, when a vehicle moving on a road network reports a position update, this can be considered a significant event causing the insertion of a new trajectory for the updated vehicle. An excellent summary of known index methods for moving points up to the year 2002 or so is contained in Agarwal et al [1] and the references therein. A recent paper by Ni and Ravishankar [12] contains a good overview of data structures experimentally validated for moving object indexing. The characteristics of some of the known approaches for indexing moving objects are summarized in Table 1.

Table 1 differentiates data structures based on their support for future time queries and whether or not they assume that the objects being indexed are constrained to move on an underlying graph. If movement is restricted to a graph, then the index should be able to take advantage of this to achieve less storage than would be required if objects were free to move anywhere in space. As we will show, the minGStree is designed to exploit this constraint.

Recently, the MON-tree [7] and PPFI [9] data structures combine R-trees to index moving objects on a fixed network. In these approaches, the network is indexed first in an R-tree. Moving objects are indexed on a forest of R-trees, whose roots are linked to leaf nodes of the network tree. Our data structure utilizes strip trees[3] to index the network, and interval trees to efficiently index moving objects.

Table 1: Different data structures for indexing moving objects. Here H = support for queries on the history of moving objects, F = support for future time queries, C = movement constrained to a planar graph, L = movement constrained to be piecewise linear, E = experimental validation.

Name	H?	F?	C?	L?	E?	
TPR-tree	N	Y	N	Y	Y	[13]
partition tree	N	N	N	Y	N	[1]
kinetic range tree	N	Y	N	N	N	[1]
MON-tree	Y	N	Y	Y	Y	[6]
MVR-tree	Y	N	Y	Y	Y	[10]
PA-tree	Y	N	N	N	Y	[12]
minGStree	Y	N	Y	Y	N	this paper

To our knowledge, our research is the first to explore the theoretical performance of data structures indexing objects moving on a graph.

4 The Primary Data Structure

We assume that N objects are constrained to move on a graph G with E edges and V vertices. The moving objects are updated when they enter or leave edges. They can begin and stop in the middle of edges. A moving object can be represented multiple times on each edge as it leaves and enters edges on its travels. n is the total number of moving object instances (i.e., time intervals on edges) of N moving objects on the entire graph over the time domain $[0, T]$.

The minGStree is a combination of strip trees and interval trees. The static strip trees are used for spatial indexing of the graph edges, and are assumed to fit in main memory. Each strip tree (at leaf level) represents a polyline (e.g., a road). The dynamic interval trees are used to index the trajectories of objects moving on the graph, and stored in external memory (as external interval trees [2]). Figure 1 illustrates a graph with $V = E = 4$. Figure 2 illustrates the corresponding minGStree arising from the graph in Figure 1.

The static part of the minGStree is based on the strip tree, but it is generalized to allow for indexing collections of strip trees representing a graph. It is a binary tree as each interior node M_i has at most two children. The tree is constructed such that interior nodes have one or two children, and such that the tree is height balanced. Besides pointing to a strip tree indexing an edge, each leaf nodes C_i of this binary tree also points to a time interval tree T_i indexing time intervals and a position interval tree P_i indexing position intervals of the moving objects on edge e_i.

A moving object instance o_ℓ^j on each edge e_i is described by a time interval $[t_1^j, t_2^j]$

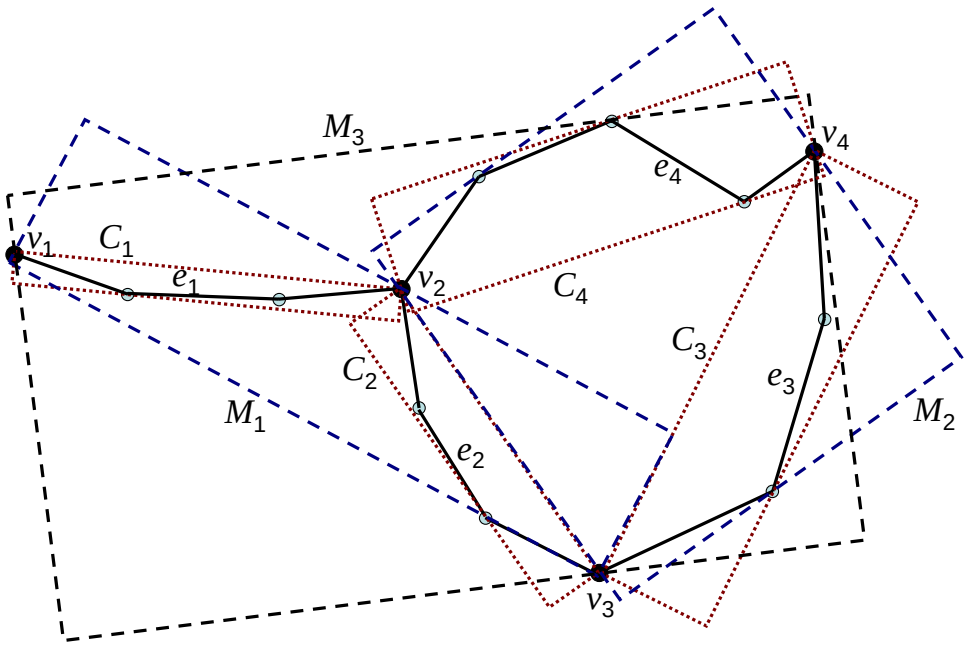

Figure 1: An example graph G with 4 edges $e_1, ..., e_4$ and 4 vertices $v_1, ..., v_4$. The edges are represented as strip trees, with $C_1, ..., C_4$ representing the root bounding boxes for each strip tree. The strip trees are merged bottom up in pairs to construct the minGStree.

and a position interval $[r_1^j, r_2^j]$. We assume that the subscript $\ell \in [1, .., N]$ refers to a unique object identifier, and the superscript $j \in [1, .., n_\ell]$ refers to a unique instance of the moving object o_ℓ at some continuous time interval $[t_1^j, t_2^j]$. We have $n = \sum_{\ell=1}^{N} n_\ell$, where n_ℓ is the total number of instances of object o_ℓ moving on edges. Figure 3 illustrates six

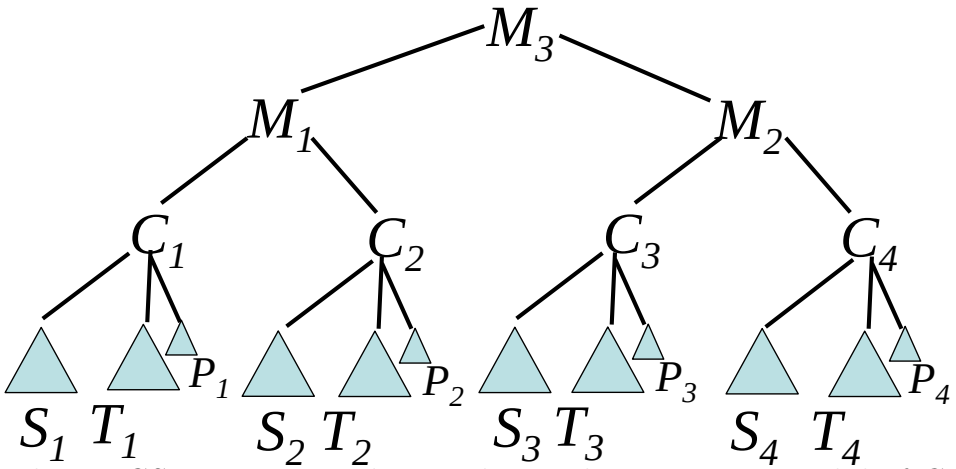

Figure 2: The minGStree corresponding to the graph in Figure 1. Each leaf C_i points to the strip tree S_i representing edge e_i, as well as to two corresponding interval trees: time interval tree T_i and position interval tree P_i.

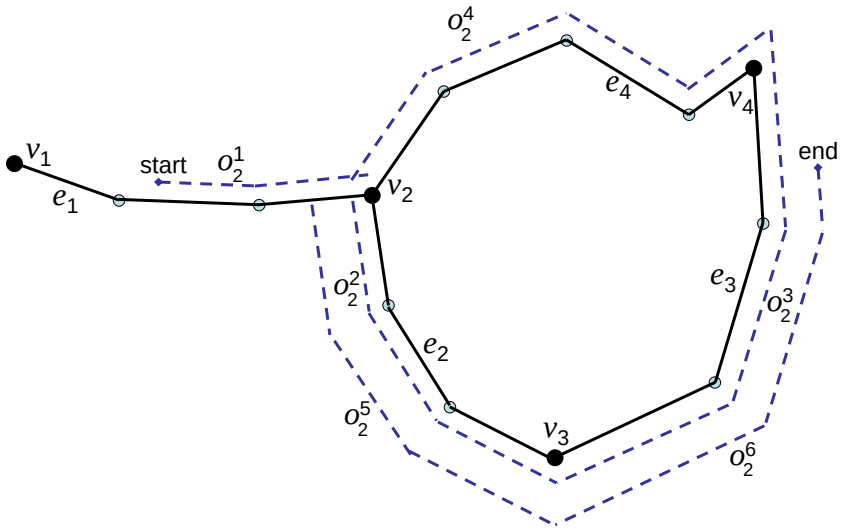

Figure 3: Six moving object instances $o_2^1, ..., o_2^6$ of o_2 moving on the graph from Figure 1.

instances of o_2 moving on the graph from Figure 1. Moving object instances o_ℓ^j are defined as objects appearing on and disappearing from edge e_i, with corresponding updates to P_i and T_i. Position intervals $[r_1^j, r_2^j]$ are often [0, 1] (i.e., moving over the entire edge) during a time interval of $[t_1^j, t_2^j]$. We use each time interval tree T_i to index time intervals of moving objects on each edge, and use each position interval tree P_i to index position intervals of these objects. Note that on a position interval tree, full intervals [0, 1] (i.e., those spanning the entire edge) do not need to be indexed. The number of intervals in P_i is much less than the number in T_i. For each moving object instance o_ℓ^j stored in T_i, we record if o_ℓ^j has position interval $[0, 1]$ or not.

Theorem 1. *For a graph containing E edges, and containing n moving object instances over the time domain $[0, T]$, the space required for the minGStree is $\Theta(E)$ memory cells and $\Theta(\frac{n}{B})$ disk blocks, for B the number of elements transmitted by one external memory access.*

Proof. There are E nodes C_i, each requiring constant space. Each internal node M_i requires constant space, and there are $O(E)$ of them, so nodes M_i require $O(E)$ space. Each strip tree S_i is a balanced binary tree, with the number of leaves depending on the resolution of the strip tree, the degree of curvature of the underlying polyline comprising the graph edge, and the number of line segments making up the polyline. We assume here that there are a constant number of leaves in each strip tree, which means that all strip trees $S_i, i = 1, ..., E$ require $O(E)$ space. There are two pointers from nodes C_i to T_i and P_i. For c the number of bytes per pointer, the space for these pointers is cE. Therefore, the total space required for the minGStree is $O(E)$ memory cells.

Since all n moving object instances are distributed over E edges, they are indexed by E external time interval trees $T_1, .., T_E$. An external interval tree requires $O(\frac{g}{B})$ disk

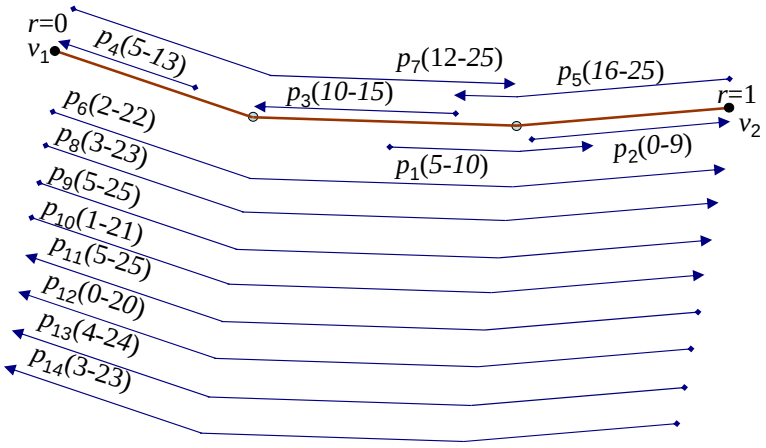

Figure 4: An example of 14 moving objects on edge e_1 of the graph in Figure 1. There are 6 moving objects having their begin or end points falling inside the edge. Eight moving objects $p_6, p_8, ..., p_{14}$ move across the entire edge. Each directed polyline represents a position interval of a moving object with a direction. The two numbers in parentheses represent time intervals of corresponding moving objects. We assume that moving objects on the same edge have the same velocity.

blocks to store g intervals (from Theorem 4.1 of [2]). The number of disk blocks used by the E time interval trees is $O(\frac{g_1}{B}) + O(\frac{g_2}{B}) + .. + O(\frac{g_E}{B}) = O(\frac{g_1+g_2+..+g_E}{B}) = O(\frac{n}{B})$, where g_i, $i = 1, .., E$ is the number of moving object instances on T_i.

There are at most two position intervals $\neq [0, 1]$ for each of N moving objects. We have at most $2N$ moving object instances stored in E position interval trees. The number of disk blocks required to store E position interval trees is $O(\frac{N}{B}) \ll O(\frac{n}{B})$. Therefore, the total $2E$ external trees in the minGStree require $O(\frac{n}{B})$ disk blocks. □

Figure 4 shows an example of moving objects for edge e_1 in Figure 1. Figure 5 illustrates how to index these moving objects on the time interval tree T_1 and position interval tree P_1. Note how each moving object has at most one interval stored for each unique time interval completely (or partially) spanning the edge. This is the minimum amount of information required to determine when a moving object travels on this edge. Along with the $O(E)$ memory cells required to store the graph structure, we can see that for graph-based moving object data structures, the space required for a minGStree is optimal.

5 Searching in a minGStree

To answer a time interval query $(R, [t_1, t_2])$, the strip trees S_i in the static part of the minGStree are used to find edges intersecting R. If an edge e_i does intersect R, a list F_i of intersecting intervals between e_i and R is constructed. Time interval trees T_i are

Obj. id	Time Int.	Position Int.
p1	5 10	0.5 0.8
p2	0 9	0.7 1
p3	10 15	0.4 0.6
p4	5 13	0 0.3
p5	16 25	0.6 1
p6	2 22	0 1
P7	12 25	0 0.68
p8	3 23	0 1
p9	5 25	0 1
p10	1 21	0 1
p11	5 25	0 1
p12	0 20	0 1
p13	4 24	0 1
p14	3 23	0 1

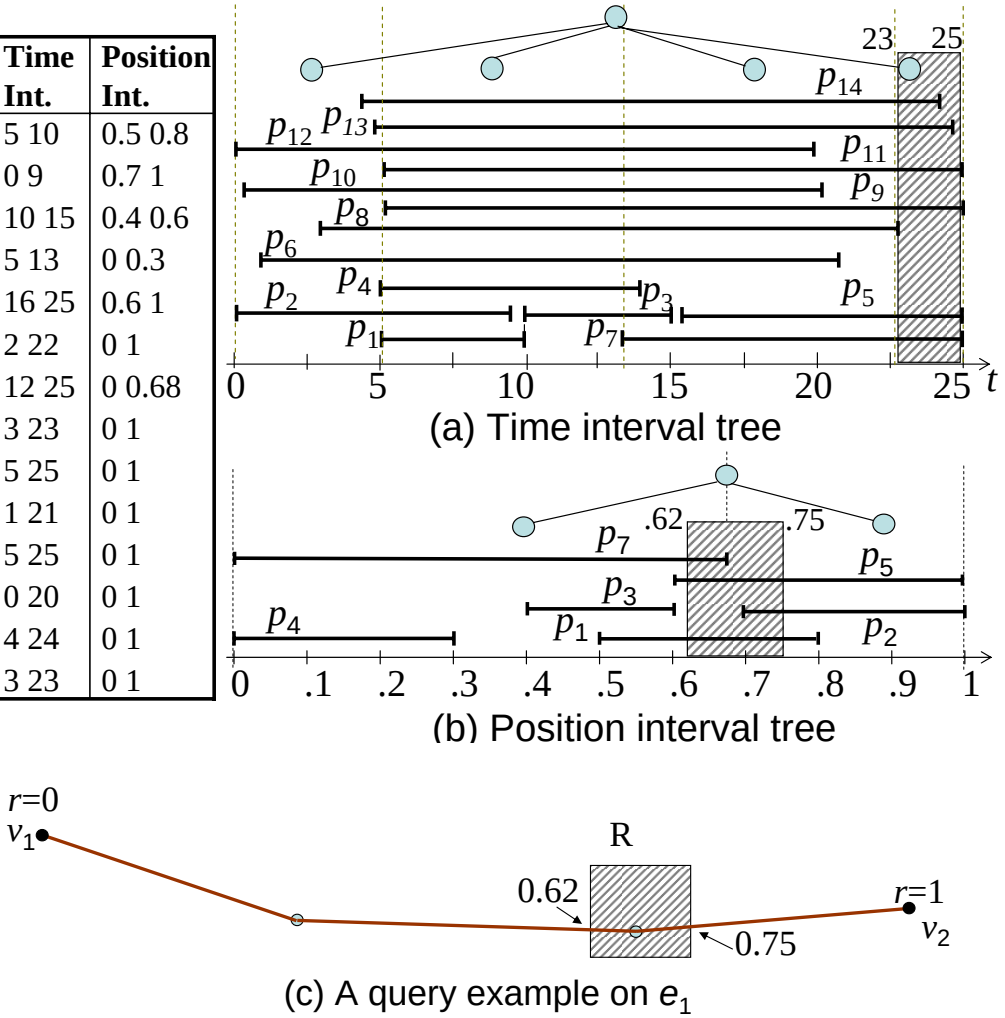

Figure 5: An example of time interval tree T_1 (a) and position interval tree P_1 (b) for e_1 of the graph in Figure 1. The table on the left shows time intervals and position intervals of 14 moving objects on e_1. There are 14 time intervals of 14 moving objects indexed in T_1, and there are only 6 position intervals of 14 moving objects indexed in P_1 since the other 8 moving objects, moving entire the edge, do not need to be indexed. The dashed vertical lines indicate slab boundaries for an external interval tree with B=7. (c) shows an example of a Q_2 query, where rectangle R intersects edge e_1 in Figure 1 at the position interval $[r_1, r_2]$=[0.62, 0.75] and the time interval query is $[t_1, t_2]$=[23, 25]. The results obtained by the search algorithm in Figure 7 are as follows: $L_1 = \{p_{14}, p_{13}, p_{11}, p_9, p_8, p_5, p_7\}$, $L_1^1 = \{p_{14}, p_{13}, p_{11}, p_9, p_8\}$, $L_1^2 = \{p_5, p_7\}$, $L_2 = \{p_7, p_5, p_2, p_1\}$, and $L = \{p_7, p_5\}$.

then used to find objects intersecting in time. With the query time interval $[t_1, t_2]$, a time interval tree returns a list L of moving objects having time intervals intersecting $[t_1, t_2]$. Position interval trees P_i are then used to further prune this list to those intersecting R.

Objects in L are in range if their position intervals in P_i intersect at least one of the query position intervals in lists F_i.

In the theorem below, we assume that the minGStree "upper part" (i.e. all except the external interval trees) can be stored in main memory. Thus, no I/Os are required to search precisely which strip trees S_i are intersected, and where.

Objects are moving continuously. The number of objects entering an edge at one time (instant) or leaving an edge at one time is constrained by edge capacity. We assume that only one moving object can enter and leave an edge at a time. In a time interval tree, moving object instances o_ℓ^j are unique.

Lemma 1. *For a Q_1=(R, t_q) query, if the position interval of an moving object instance o_ℓ^j is $[0, 1]$, $R \cap e_i \neq \emptyset$, and $t_1^j \leq t_q \leq t_2^j$ in the time interval tree T_i, then o_ℓ^j satisfies Q_1.*

Proof. For $[r_1^j, r_2^j]$= $[0, 1]$, the object o_ℓ^j moves across the entire edge, so o_ℓ^j intersects R if $R \cap e_i \neq \emptyset$. All that remains is to check if the query time t_q intersects the object time interval $[t_1^j, t_2^j]$. □

A similar lemma holds for a query type Q_2.

Lemma 2. *The expected number of intervals in a time interval tree T_i is $O(\frac{n}{E})$.*

Proof. We assume objects are moving at a constant velocity $v \in [v_{min}, v_{max}]$ per edge. Initially, N moving objects are distributed in a uniform random fashion over the E edges at a rate of η per unit distance. The number of moving object instances flowing through short edges is equal to the number of moving object instances flowing through long edges over the time interval $[0, T]$. For n moving object instances, as time $t \rightarrow T$, where $T > max(u_i)/v_{min}$, u_i is the length of edge e_i, the number of moving object instances per edge $\rightarrow \frac{n}{E}$. Thus, each time interval tree T_i stores $O(\frac{n}{E})$ time intervals, one for each moving object instance. □

Lemma 3. *The expected number of intervals in a position interval tree P_i is $O(\frac{N}{E})$.*

Proof. The number of moving object instances in P_i is the number of position intervals $\neq$ [0,1] on edge e_i. There are N unique moving objects on a graph with E edges. As an object moves over time from one edge to another, only two end points exist (for the path of one object in the graph). At most two moving object instances with position intervals $\neq$ [0,1] can arise from each moving object, where 0 or 1 corresponds to vertex locations. There are N objects moving on E edges, so we expect at most $\frac{2N}{E}$ (or $O(\frac{N}{E})$) moving object instances with position intervals $\neq$ [0,1] in each tree P_i. □

Lemma 4. *On an interval tree containing g intervals, performing a range query $[t_1, t_2]$ takes $O(\log_B g+k)$ I/Os, where k is the number of disk blocks required to store the answer.*

Proof. From Theorem 4.1 of [2], a stabbing query on a interval tree containing g intervals requires $O(\log_B g + k')$ I/Os, where k' is the number of disk blocks required to store the answer. A range query requires I/Os for two stabbing queries at t_1 and t_2, plus I/Os for intersecting intervals stored in tree nodes in the *'middle part'*. We call the two leaf nodes that stabbing queries at t_1 and t_2 reach β_L and β_R, respectively. Assume β is the nearest common ancestor of β_L and β_R. The *'middle part'* contains all nodes of the interval tree that fall between the two paths from β to β_L and β to β_R (see Figure 6). All intervals stored in the *'middle part'* nodes intersect $[t_1, t_2]$, which require reading all the *'middle part'* nodes from disk. If the stabbing query at t_1 requires $O(\log_B g + k_1)$ I/Os, the stabbing query at t_2 requires $O(\log_B g + k_2)$ I/Os, and reading the *'middle part'* nodes requires k_3 I/Os, the number of I/Os required for a range query is $O(\log_B g + k_1 + \log_B g + k_2 + k_3)$ $= O(\log_B g + k)$, where k is the number of disk blocks required to store the answer. □

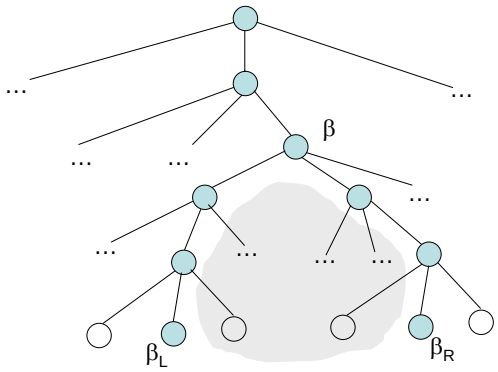

Figure 6: Illustration of external interval tree nodes in a range query $[t_1, t_2]$. β_L and β_R are the two leaf nodes that two stabbing queries at t_1 and t_2 reach. β is the nearest common ancestor of β_L and β_R. The shaded area contains nodes in the *'middle part'* between two stabbing query paths β to β_L and β to β_R.

Theorem 2. *For n moving object instances randomly distributed on the E edges of a graph, the number of I/Os required to determine the moving objects intersecting one edge at time t_q (or at time interval $[t_{q1}, t_{q2}]$) in a minGStree is expected to be $O(\log_B \frac{n}{E} + k)$, where k is the number of disk blocks required to store the answer.*

Proof. From Lemma 1, the corresponding time interval tree T_i is used to find objects intersecting the query time t_q or time interval $[t_{q1}, t_{q2}]$. The expected number of intervals in T_i is $O(\frac{n}{E})$ (Lemma 2), and searching on T_i takes $O(\log_B \frac{n}{E} + k_1)$ I/Os (Lemma 4). The query then visits position interval tree P_i for pruning moving objects in range in time but not in positions.

As shown in Figure 7, the list of objects in range from T_i is called L_1. Since L_1 includes objects satisfying the query time, if L_1 is empty, we don't need to search on P_i. Otherwise, we divide L_1 into two parts: L_1^1 for objects with their position intervals = [0,

1], and L_1^2 with their position intervals $\neq [0,1]$. The list of objects in range from P_i is called L_2. Reported objects will be objects in L_1^1, and common objects from L_1^2 and L_2. As steps for dividing L_1 and finding the common objects from L_1^2 and L_2 are performed in main memory, the number of I/Os required comes solely from the interval search on T_i and P_i.

From Lemma 3, the number of intervals in P_i is $O(\frac{N}{E})$, so searching on P_i requires $O(\log_B \frac{N}{E} + k_2)$ I/Os. Therefore, a Q_1 or Q_2 query on one edge of a minGStree requires $O(\log_B \frac{n}{E} + \log_B \frac{N}{E} + k_1 + k_2)$, or $O(\log_B \frac{n}{E} + k)$ I/Os (since $n \gg N$). □

Figure 5(c) gives an example of a query rectangle R intersecting edge e_1 and also illustrates how objects in range of e_1 are computed based on the searching algorithm in Figure 7.

Algorithm 1: Searching(T_i, P_i, R, t_1, t_2)

The general algorithm for time interval searching on one edge e_i.

input : Root node of T_i, root node of P_i, query time interval $[t_1, t_2]$, and R: list of query intervals on edge e_i
output: a list L of moving objects in range

```
1.1  begin
1.2      L ← ∅
1.3      L1 ← intervalSearch(Ti, t1, t2)
1.4      if L1 ≠ NULL then
1.5          L1^1 ← entries in L1 with [r1, r2] = [0, 1]
1.6          L1^2 ← L1 − L1^1
1.7          L2 ← intervalSearch(Pi, R)
1.8          L ← L1^1 ∪ (L1^2 ∩ L2)
1.9      return L;
1.10 end
```

Figure 7: The general algorithm for time interval searching on one edge e_i.

Theorem 3. *For n moving object instances randomly distributed on the E edges of a planar graph, the number of I/Os required to determine the moving objects intersecting a Q_2 query in a minGStree is expected to be $O(\log_B \frac{n}{E} + k)$, where k is the number of disk blocks required to store the answer.*

Proof. Strip trees S_i, stored in the main memory, are used to determine if R intersects an edge e_i. This step takes zero I/Os. From Theorem 2 searching on one edge requires $O(\log_B \frac{n}{E} + k)$ I/Os. If R of Q_2 intersects D edges of the graph, searching on D edges takes $O(D \log_B \frac{n}{E} + k)$. Assuming D is a constant, much smaller than E, the expected number of I/Os required to answer a Q_2 query is $O(\log_B \frac{n}{E} + k)$, where k is the number of disk blocks required to store the answer. □

In the worst case, when R intersects all E edges, the number of I/Os required to determine moving objects falling in range is $O(E \log_B \frac{n}{E} + k)$. However, in the next corollary, we will show that this bound is dominated by k.

Corollary 4. *When rectangle R intersects all E edges and the time domain of the minGStree $[0, T] \subset [t_1, t_2]$, the number of I/Os required to answer this query is $O(E \log_B \frac{n}{E} + k)$. This expression is dominated by k, where k is the number of disk blocks required to store the answer.*

Proof. Assume at most E^c moving objects N on the graph, for c a small constant and greater than 1. Assume each moving object travels on $O(E)$ edges. Thus, the number of moving object instances $n = O(E^{c+1})$. If all moving object instances of one edge e_i fall in range, we expect $k_i = \frac{n}{EB}$. For R intersecting E edges, the number of I/Os expected is $E \times O(\log_B \frac{n}{E} + k_i) = O(E \log_B \frac{n}{E} + E \times \frac{n}{EB}) = O(E \log_B \frac{n}{E} + \frac{n}{B}) = O(E \log_B \frac{E^{c+1}}{E} + \frac{E^{c+1}}{B})$ $= O(E \log_B E^c + \frac{E^{c+1}}{B}) = O(E(\log_B E + \frac{E^c}{B}))$. For large graphs, we assume B is always smaller than E, so $\log_B E$ is always smaller than $\frac{E^c}{B}$, or $E \log_B E$ is always smaller than $\frac{E^{c+1}}{B}$. Therefore, the number of I/Os k dominates the expression $O(E \log_B \frac{n}{E} + k)$.

For reasonable assumptions about the size of N and E, the number of I/Os required in the worst case is dominated by k, the number of blocks of external memory required to store the output.

□

Corollary 5. *Inserting or deleting an moving object instance into or from the minGStree is expected to require $O(log_B \frac{n}{E})$ I/Os amortized.*

Proof. Insertion or deletion of an moving object instance in the minGStree happens in one time interval tree T_i and one position interval tree P_i. From [2], we know that the amortized I/Os required for updating an I/O interval tree are $O(log_B \frac{n}{E})$. □

For $R \cap G = \emptyset$, or $[t_1, t_2] \cap [0, T] = \emptyset$, we can trivially determine that the query answer is $\emptyset$ with 0 I/Os.

6 Conclusion

We present a new data structure, the minGStree, for efficient search of moving objects (e.g. vehicles) on planar (or non-planar) graphs. The minGStree is a combination of strip trees and interval 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). The interval trees are used to index the trajectories of moving objects on graphs indexed by strip trees. There are some advantages for the minGStree. First, the top strip trees and the bottom interval trees are independent. For example, one can update interval trees without changing the strip tree indexing for edges, or update a strip

tree when an edge changes, without affecting other strip trees (at leaf level). Second, since moving objects on a graph edge belong to a strip tree, we can easily answer queries which count moving objects on a specific edge; for example, how many vehicles move on a specific road at a specific time or during a specific time interval.

The minGStree is directly implementable as an I/O efficient data structure by replacing the internal memory interval trees with the optimal external memory interval tree. It remains to experimentally validate the minGStree with an implementation of I/O-efficient external memory interval trees. (e.g. [5]).

7 Acknowledgements

This research is 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] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. *Journal of Computer and System Sciences*, 66:207–243, 2003.

[2] L. Arge and J. S. Vitter. Optimal external memory interval management. *SIAM J. Comput.*, 32(6):1488–1508, 2003.

[3] D. H. Ballard. Strip trees: a hierarchical representation for curves. *Communications of ACM*, 24(5):310–321, 1981.

[4] J. Basch, L. J. Guibas, and J. Hershberger. Data Structures for Mobile Data. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 747–756, New Orleans, Louisiana, US, 5-7 January, 1997.

[5] Y.-J. Chiang and C. T. Silva. External memory techniques for isosurface extraction in scientific visualization. In *in External Memory Algorithms and Visualization, DIMACS Series in Discrete Mathematics and Theoret. Comput. Science 50*, pages 247–277, 1999.

[6] V. T. de Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.

[7] V. T. de Almeida and R. H. Güting. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.

[8] 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.

[9] 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.

[10] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Indexing spatiotemporal archives. *VLDB Journal*, 15(2):143–164, 2006.

[11] 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.

[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] 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.