SPATIAL AND MOVING OBJECT DATABASES [CIS 6930]

REPORT

ANKIT RAMESHWAR SHARMA

UF ID : 24868901

TOPIC # 23

DATA STRUCTURES FOR MOVING OBJECTS

Table of Content

Abstract

The main aim of this paper is to provide a Data Structure to support queries and perform operations on time dependent geometric data stored as Spatio-temporal(Moving) data. The data of moving object is stored with the time dependent position of object and manipulated. To come up with a data structure we first come up with a data model involving complex spatial structures and operations, which can be plugged into DBMS, which support query language. A concept of sliced representation is introduced, it represents temporal development of moving objects as a set of units, temporal development within each such unit can be represented by simple linear function of time. Then an algorithm describing the use and implementation of the data structure and its complexity is discussed. Other data structures which solve the challenges faced by moving objects are also discussed.

Introduction

Moving Object Databases

The name itself is self explanatory. Moving object is an object changing its geometry with time. Hence Moving Object Database is the Database dealing with these moving objects. Actually Moving Objects Database are nothing but spatial databases (Spatial Database is full-fledged database that enables us to store, retrieve, manipulate and query geometries like point, lines and regions) which not only deals with these changing geometries but also concerns with their change in location, shape and extent over time. In addition it offers Spatiotemporal Data Types (STDTs) for moving objects (including operations and predicates) in its data model and query language. It supports STDTs by providing Spatiotemporal indexing and efficient algorithms for Spatiotemporal joins.

i. Applications and examples :

Moving Objects are ubiquitous, they are used in various applications.

1. Geographic resource discovery: A mobile user provides its partial future trajectory to a service provider, and expects the answer to such queries as, "notify me when I am two miles away from a motel that has rooms available for under $50 per day." The service provider uses a MOD to store the location information of its customers and answer their queries/triggers.

2. Transportation (taxi, courier, emergency response, municipal transportation, traffic control, supply chain management, logistics): In these applications the MOD stores the trajectories of the moving objects and answers such queries as: "which taxi cab is expected to be closest to 320 State Street half an hour from now" (when presumably service is requested at that address); "When will the bus arrive at the State and Oak

station?" "How many times during the last month was bus #25 late at some station by more than 10 minutes?"

3. Mobile workforce management**:** Utilities and residential/commercial service providers track their service engineers and the MOD answers such queries as: "Which service crew is closest to the emergency at 232 Hill Street?"

4. Air traffic control**:** Currently commercial flights take "highways in the sky," but when free-flight (FAA, 2004) is instituted, a typical trigger to the air-traffic control MOD may be: "Retrieve the pair of aircraft that are on a collision course, that is, are expected to be less than a mile apart at some point."

Figures below show pictorial representation of moving objects viz. All routes of airplanes and migratory route of whales.

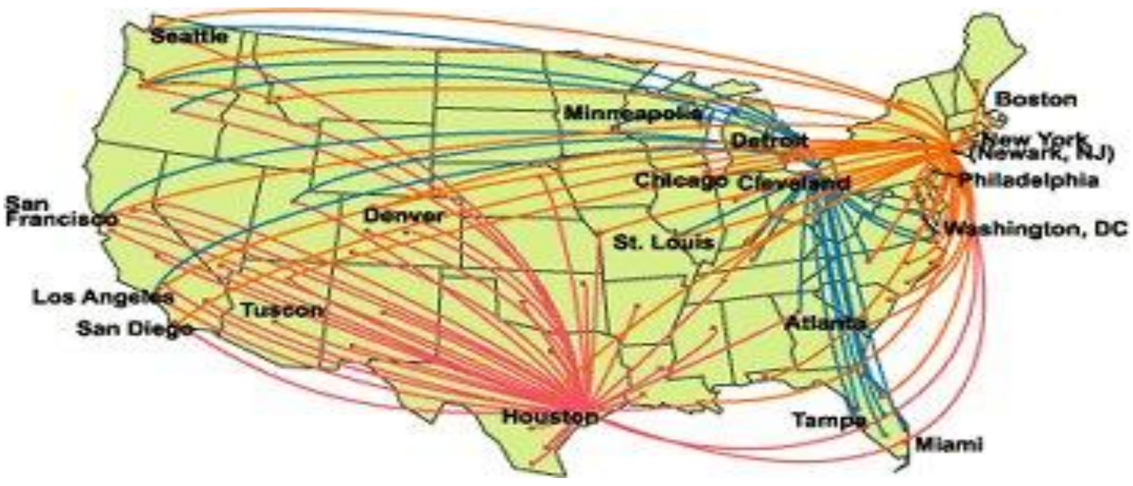


Fig1. Path of Airplanes

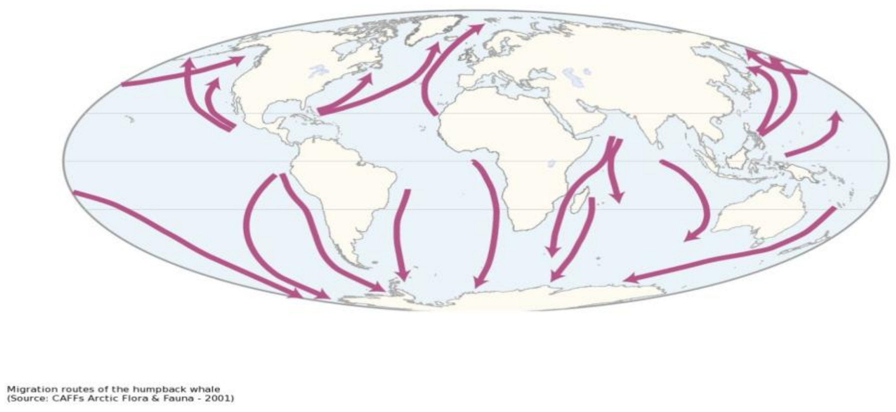

Migration routes of the humpback whale
(Source: CAFFs Arctic Flora & Fauna - 2001)

Fig2. Migration of Humpback Whales

Hence a wide and range of database applications deal with such moving objects and they are indeed ubiquitous.

Data Modelling

Modelling is done to give a high level description of which datatypes and operations are to be used. For modelling fundamental abstractions are points, line and region. These geometries change over time. Point is just a single point in Euclidean plane and Points is a multiple point, they represent an object whose position and not extent is relevant (eg. City on map). A line is a set of curve in space and it describes facilities for moving through space or connections in space (road, rivers, power lines). A region is just a collection of faces which occupy certain area in the plane, it represents an abstraction of the object whose extent is relevant eg. Forest or lake. We consider objects to be as sequence of these elements where it is possible to access elements consecutively and to insert a new element into this sequence and model the three data structures accordingly.

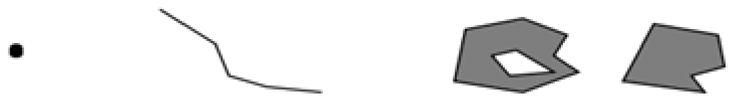
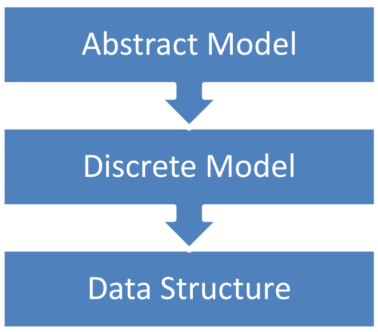


Fig3.

For modelling of Data Structure we define first Abstract Level, then Discrete Level and then finally Implementation Level. Abstract and Discrete model can be compared to interface and class respectively. Just like interface, Abstract model just gives definition in infinite domain and is not concerned with implementation details. Discrete model on the other hand is like class is concerned with implementation details and represents definitions in finite domain.

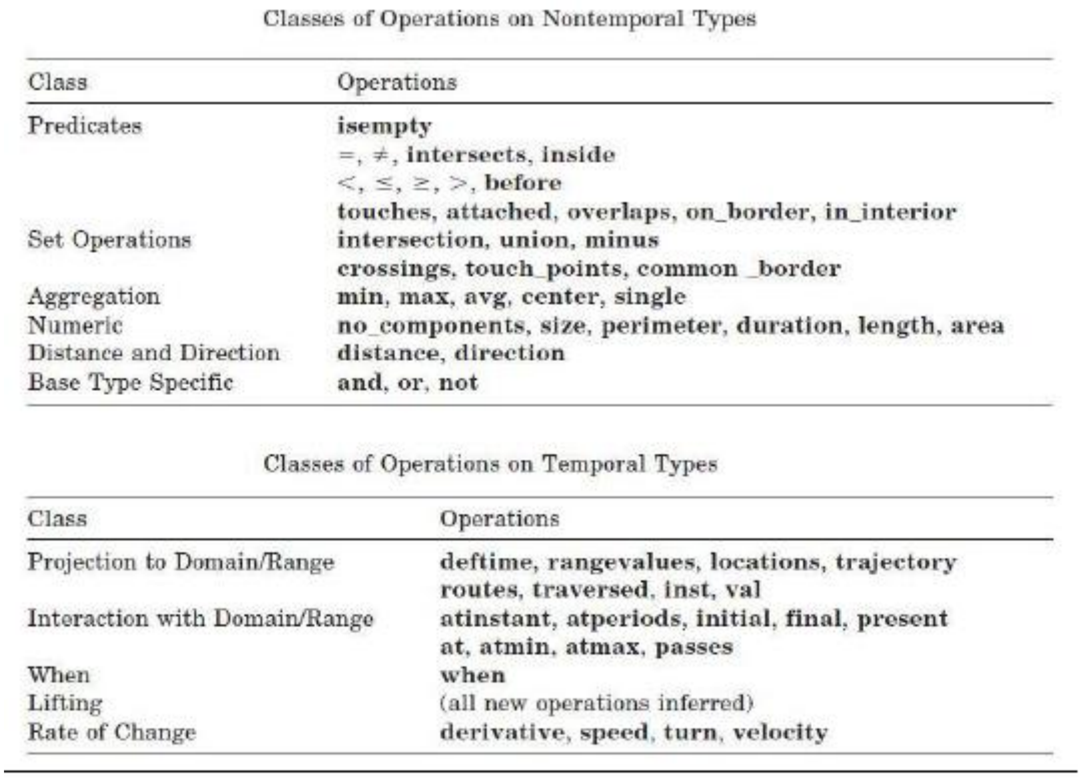


i. Abstract Model

Abstract model is a conceptual model and it gives us a high level idea of Data Structures. It defines them in terms of infinite sets, without being concerned about their finite representation. As it is not concerned with implementation, hence it is relatively clean and simple. Drawback of this model is that as it has infinite representation it cannot be stored, manipulated and used by algorithms. It tells the type of data stored in the Data Structure and the operations allowed.

Operations

We need to provide operations to these Data Types following the principle of generality and consistency. Operations should be general i.e. in one dimension int and in two dimension range(int) are related to make them scale in two dimensions. This generality allows scaling. The operations should be supported by both temporal and non temporal data types. Lifting operation comes handy when we define operations to non temporal types and extend them to temporal types.



Classes of Operations on Nontemporal Types

Class	Operations
Predicates	**isempty**
	=, ≠, **intersects**, **inside**
	<, ≤, ≥, >, **before**
	touches, **attached**, **overlaps**, **on_border**, **in_interior**
Set Operations	**intersection**, **union**, **minus**
	crossings, **touch_points**, **common _border**
Aggregation	**min**, **max**, **avg**, **center**, **single**
Numeric	**no_components**, **size**, **perimeter**, **duration**, **length**, **area**
Distance and Direction	**distance**, **direction**
Base Type Specific	**and**, **or**, **not**

Classes of Operations on Temporal Types

Class	Operations
Projection to Domain/Range	**deftime**, **rangevalues**, **locations**, **trajectory**
	routes, **traversed**, **inst**, **val**
Interaction with Domain/Range	**atinstant**, **atperiods**, **initial**, **final**, **present**
	at, **atmin**, **atmax**, **passes**
When	**when**
Lifting	(all new operations inferred)
Rate of Change	**derivative**, **speed**, **turn**, **velocity**

ii. **Discrete Model**

Discrete model is realistic model and gives us finite representation of the infinite definitions and shapes given by Abstract model. Hence it is subset of the abstract model. It defines the domain of values which datatype can take in terms of finite representation

Signature describing the discrete type system.

	→ BASE	*int*, *real*
		string, *bool*
	→ SPATIAL	*point*, *points*
		line, *region*
	→ TIME	*instant*
BASE ∪ TIME	→ RANGE	*range*
BASE ∪ SPATIAL	→ TEMPORAL	*intime*
BASE ∪ SPATIAL	→ UNIT	*const*
	→ UNIT	*ureal*, *upoint*
		upoints, *uline*
		uregion
UNIT	→ MAPPING	*mapping*

Correspondence between abstract and discrete temporal

Abstract type	Discrete type
moving(*int*)	*mapping*(*const*(*int*))
moving(*string*)	*mapping*(*const*(*string*))
moving(*bool*)	*mapping*(*const*(*bool*))
moving(*real*)	*mapping*(*ureal*)
moving(*point*)	*mapping*(*upoint*)
moving(*points*)	*mapping*(*upoints*)
moving(*line*)	*mapping*(*uline*)
moving(*region*)	*mapping*(*uregion*)

As this model is concerned with implementation and representation details, it is more complex. It gives us basis for the implementation of Data Structures. Also it can be used in algorithms.

Both models are necessary. Abstract model gives simple and plain understanding of concepts and design of query operations whereas discrete model gives finite domain implementation.

Data Structures

We face challenges with the Moving Object Database such as efficient storage and retrieval of the data. General technique for storing such moving objects is taking snapshots at various points in time, though this represents the various state well but its limitation is that it misses out on the events that lead to these changes. Also the space requirement is quiet high as object is to be replicated at every point in time.

Hence we need Data Structure to solve these problems, keep track of moving objects and store them in such a way that queries on them can be answered quickly. So a data structure provides access to these stored moving objects in an organized manner and allows analyzing and querying on moving objects. Data Structure provides systematic way of organizing and accessing data. They bundle data with operations that manipulate the data. They have operations to access, insert and remove them from collection.

Data Structures are to be implemented within a database system and hence we need to make sure certain points. First, as the values of Data Structures are to be placed under control of DBMS, we cannot use pointers. Secondly, representation should consist of small number of memory blocks that can be moved efficiently between main and secondary and memory.

Following we discuss the three datatypes which are used to represent the geometries of moving objects viz. Point(s), line and region.

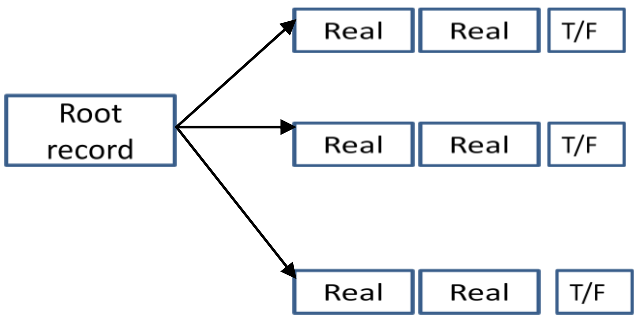
i. Points

A point is represented by x- and y-coordinates. Hence a point can be represented by an array containing two real fields, x- and y-coordinate, and a Boolean flag, indicating whether the flag is defined or not, to represent a point.

. P(x,y)

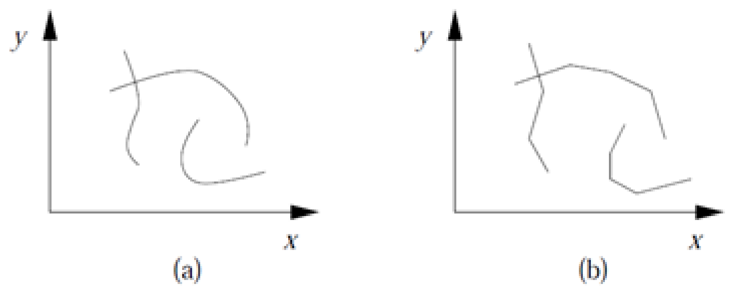
Fig. A point in space

Points contains more than one points and is represented as below. The root record holds reference to the database array and the number of points.



ii. **Line**

A line is a continuous curve in 2D plane. This curve is represented as a line segments (fig 2). Hence a line is a set of half line segments. Each such segment in turn is represented by two half segments, one for the left end point and other for the right. This set of line segment is stored in an array containing the ordered sequence of halfsegments records.



The purpose of this representation is to support plane-sweep algorithms, which traverse a set of segments from left to right and have to perform actions, such as insertion into sweep status structure, on encountering the left or right end points of a segment.

A line is represented by half segments, each half segment has two end points, and each point needs two reals to be represented. Each half segment represents a dominating, left or right end point, point representing left and right end of the line.

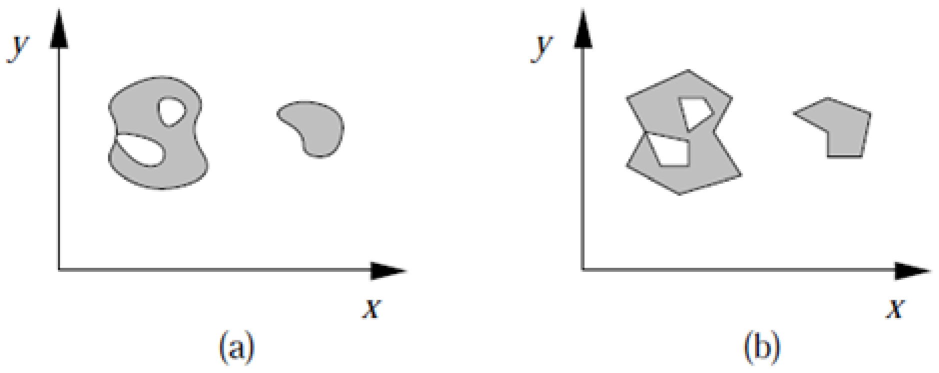
P(x,y).____. Q(x,y)

Hence a line segments is represented by four real fields and an extra field for dominating point, representing either left or right end point. The root record manages the database array and holds some auxiliary information such as the number of segments, total length of segments, bounding box etc.

Real	Real	Real	Real	Flag

iii. **Region**

A region can be viewed as a set of polygons represented by line segments with some additional data structure (fig 3). Just like line, the set of line segments representing it are given by an array of halfsegments containing the ordered sequence of halfsegments records.



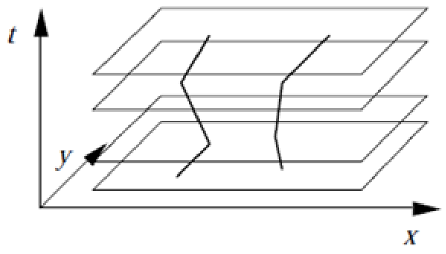
A region is represented by faces and cycles, hence two more arrays representing faces and cycle represent the data structure. Two more arrays viz. Cycles and Faces are used to represent the structure. Array cycle holds the pointer to the first half segment in it and a pointer to the next cycle, and hence links all cycles belonging to one face. All halfsegments belonging to a cycle and to a face are linked together, via extra fields such as next-in-cycle. The root record holds pointer to the database array and stores additional information such as bounding box, number of cycles, total area, perimeter etc.

Sliced Representation

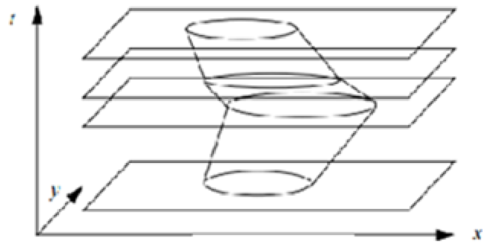
It is representation of temporal development of moving objects with a sequence of timely ordered slices, also called as units. The basic idea is to break down the temporal development into smaller parts, each such part is represented by slice. Important property of sliced representation is that within each such slice the temporal development can be represented by simple linear function. Diagram below shows sliced representation of moving point and a moving region in 3D space.

Slices may or may not be at the uniform distance, but they represent simple function. Another property of slice is that they represent disjoint time intervals and if they are adjacent then their time intervals are unique. The below equation denotes it, Mapping is the constructor which gives Sliced Representation.

$$\begin{aligned}
&Mapping(S) = \{U \subseteq Unit(S) \mid \forall(i_1, v_1) \in U, \forall(i_2, v_2) \in U : \\
&\quad (i) \quad i_1 = i_2 \Rightarrow v_1 = v_2 \\
&\quad (ii) \quad i_1 \neq i_2 \Rightarrow (disjoint(i_1, i_2) \wedge (adjacent(i_1, i_2) \Rightarrow v_1 \neq v_2))\}
\end{aligned}$$



Sliced representation of moving points value



Sliced representation a moving region representation

i. Operations

The first set of operations is provided for manipulating moving points. The get first slice operation retrieves the first slice unit in a slice sequence of a moving point, and sets the current position to 1. The get next slice operation returns the next slice unit of the current position in the sequence and increments the current position. The predicate end of sequence yields true if the current position exceeds the end of the slice sequence. The operation create new creates an empty MPoint object with an empty slice sequence. Finally, the operation add slice adds a slice unit to the end of the slice sequence of a moving point.

The second set of operations is provided for accessing elements in a slice unit. The operation get interval returns the time interval of a slice unit. The operation get unit function returns a record that represents the linear function of a slice unit. The create slice operation creates a slice unit based on the provided time interval and the linear function.

ii. Algorithm

The following algorithm illustrates the use of data model and data structure. The below algorithm defines the Boolean predicate, returning true or false, describing if the moving point is inside a moving region or not and if yes then when.

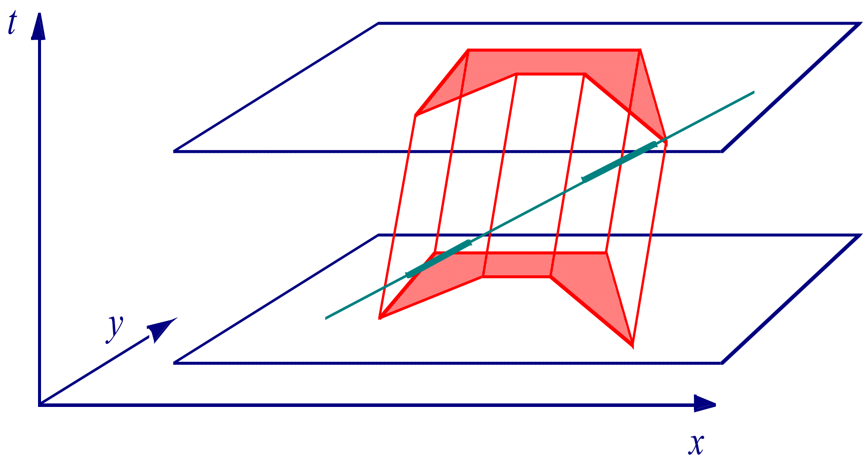
Input : two array lists of units, one representing moving point mp (of type mapping (upoint)) and other representing moving region (of type mapping (uregion)).

Output : A moving Boolean mb (of type mapping const(bool)), representing when mp was inside mr.

```
inside(mp,mr)
        mp={up1, up2 …}    mr={ur1,ur2…}
        mb=null
            for each time interval i
          ub = upoint_uregion_inside(up, ur)
          mb=concat(mb,ub)
      endfor
return mb
```

```
upoint_uregion_inside(up, ur)
let up = (i', mpo) and ur = (i'', F) and i = [s, e]
        the intersection time interval;
find all k intersections, sort them by time;

if k = 0
        if mpo at instant s inside F at instant s
        then return {([s, e], true)}
        else return {([s, e], false)}
        endif
```



```
else let the list of intersections be
        $(t_1, a_1), \ldots, (t_k, a_k)$
        where $a_i \in$ {enter, leave}
        if $a_1$ = leave
        then return… (alternating intervals, starting with true)
        else return … (alternating intervals, starting with false)
        endif
endif
```

iii. <u>Analysis/Complexity</u>

The moving point object (mp) is a line segment in 3D that may stab moving segments, which are trapeziums in 3D , of F. Each intersection of moving point alternates between entering and leaving the moving region unit. So a list of boolean units is produced which alternates between true (enters region) and false (leaves region). If there are no intersections then we will find if during the start time point was inside the region and return true and false if otherwise.

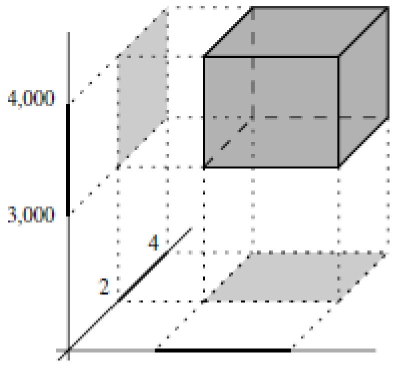
The first function requires O(p+r) time, p is number of units in moving point and r is number of units in moving region. The second method requires O(S) time for finding all intersections, S is number of msegments in moving region. If there are k intersections between moving point and moving region, then we need O(k logk) time to sort. If there are no intersections then we need O(S) time to check if the point was inside the moving region. Hence total time is O(S + k logk) to make all calls to function upoint_uregion_inside. In practice number of intersection (k) will be less and hence term k logk will be dominated by S, so total running time of Algorithm can be represented as O(p+r+S).

More Data Structure

We face more challenges such as nearest neighbour search, Range searching on a grid, the point location problem, computation of rectangle intersections and maximal elements by divide-and-conquer and plane sweep algorithm, computing the convex hull of a set of points and reporting all intersections of a set of arbitrarily oriented line segments can be answered with the following data structures.

Orthogonal Range Query Searching :

Find employees who are aged between 35 and 40, and their salary between $60,000 a nd $80,000 and have either 2 or 3 children



35 40

Above we are querying on three fields of records so we convert points into 3-dimensional space. For d- fields we will transform it into d-dimensional space. Such a query asking to report all records whose fields lie between specified values then transforms to query asking for all points inside a d-dimensional axis parallel box. Such a query is called a rectangular or orthogonal range query.

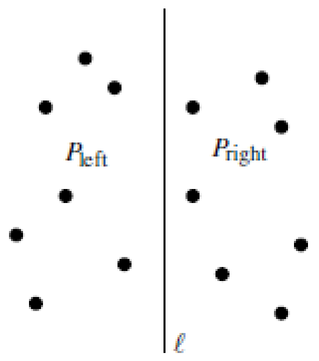
1-Dimensional Trees

The data we are given is a set of points inside a 1-Dimensional space eg. Set of real numbers. A query asks for points inside a 1-dimensional query rectangle or interval [start,end]. The 1-dimensional range searching problem can be solved using arrays but this solution does not generalise to higher dimension problem and nor does it allows for efficient updates on set of points on given line. It can be efficiently implemented using Balanced Binary Search Tree. As it’s a balanced BST it uses O(n) storage and can be built in O(n logn) time. Query time will be O(n).

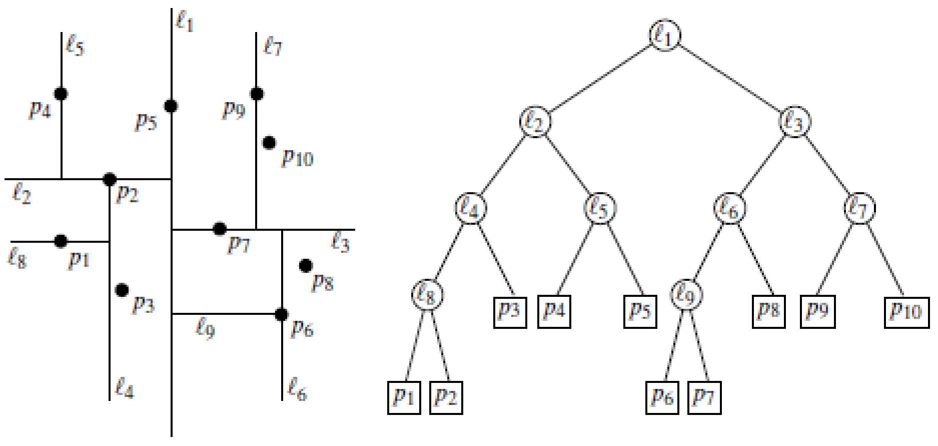
<u>K-Dimensional (Kd) Trees</u>

It answers 2-dimensional range searching problems. 2-dimensional queries is composed of two 1-Dimensional queries. Its implemented using binary search tree and hence it is splited into two halves of roughly equal size. One half contains elements smaller than root, which is the splitting value, and other half of elements bigger than the root. In 2-dimensional we first split the points on x- and then on y-coordinate and do this repetitively.

At the root we split the set P with a vertical line into two subsets of roughly equal size. The splitting line is stored at the root. Pleft, the subset of points to the left or on the splitting line, is stored in the left subtree, and Pright, the subset to the right of it, is stored in the right subtree. .



At the left child of the root we split Pleft into two subsets with a horizontal line; the points below or on it are stored in the left subtree of the left child, and the points above it are stored in the right subtree. The left child itself stores the splitting line. Similarly, the set Pright is split with a horizontal line into two subsets, which are stored in the left and right subtree of the right child. At the grandchildren of the root, we split again with a vertical line. In general, we split with a vertical line at nodes whose depth is even, and we split with a horizontal line at nodes whose depth is odd.



Above is shown a the splitting process and the corresponding binary tree, also called as Kd tree. It’s an approach relying on binary subdivision.

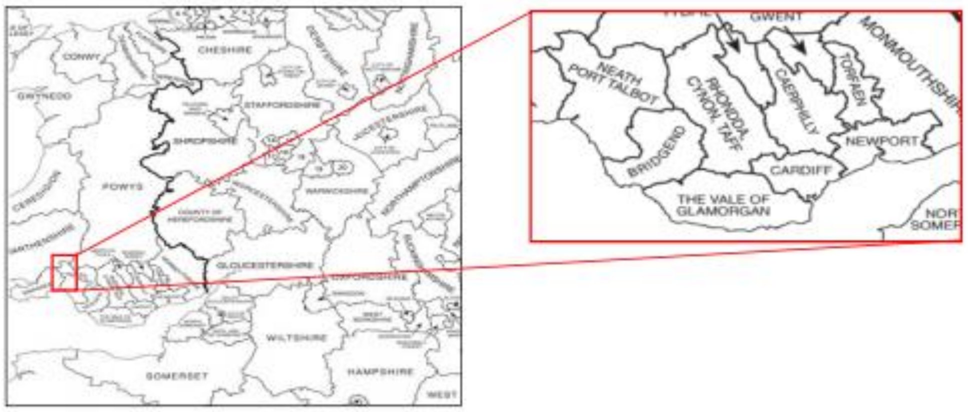
The space for storage of Kd tree is O(n) and construction time is O(n logn). The query time for k reported points is O(√n + k). Query time is high when the number of reported points are small.

Range Trees

Kd-trees, which were described in the previous section, have O(√n+k) query time. So when the number of reported points is small, the query time is relatively high. So we have another Data structure the range tree, which as a better query time, namely O(log^{2}n+k), though it can be improved to O(logn + k) by fractional cascading. The price we have to pay for this improvement is n increase in storage from O(n) for kd-trees to O(n logn) for range trees.

Window Queries

Example of Window Queries use is geographical maps. Suppose while navigating using GPS, only small portion of map such as road, city etc needs to be displayed. Considering the amount of data we have, so to check every feature of map to see if it lies within the window is not workable. So we should store the map in a data structure that allows to retrieve the part inside window quickly. Data for window querying are line segments, curves & polygons and search space usually is 2 or 3-dimensional.



Data Structures for Windowing Queries are :-

1. Interval Trees
2. Priority Search Trees

3. Segment Trees

1. **Interval Trees**

It stores the line segments intersecting the query windows in such a way that they can be reported efficiently. We identify the line segments which intersect window and the way they intersect, twice, inside and partially overlap, by applying range queries within the window. Implementation of interval trees has three versions. These versions can be implemented using binary tree or red black tree.

An implementation with binary trees uses O(n) storage, its depth is O(log n), time for query to report all such intervals(k) containing query point is O(log n + k). Implementation with red-black tree has height of O(log n), searching an interval takes O(log n) time and insert/delete takes O(log n) time.

2. **Priority Search Trees**

Priority Search Tree are much simpler as it does not need fractional cascading and also reduces the storage bound to O(n). It is implemented using binary tree/heap. They are normally used for the priority queues where smallest or largest value are required by the query.

Priority Search Tree uses improves the bound on this storage to O(n) and can be built in O(n logn) time and answer range query in O(log n + k) time, k are reported points.

3. **Segment Trees**

A concept of bounding box is used. We find all bounding box which intersect the query window and then find all the segments in the bounding box which intersect the window. In worst case all bounding box will intersect window and none of the segments intersects window. So if a fast query time is to be guaranteed then it should not be used. In practice it works quiet well as majority of the segments of the bounding box intersect window. It is implemented using binary tree.

Storage Complexity of segment tree is O(n logn), insert/delete is O(log n) and for search query it is O(log n +k).

Conclusion

This paper presented what are moving object databases and there applications. It presents abstract and discrete model that gives a detailed outline and how they can be mapped into data structures describing moving objects. It also presents the operations for these data structures following principles of generality, consistency and closure property. It presented an algorithm showing actual implementation and use of this data structure. It also presented a concept of Sliced Representation which breaks down movement of moving object into units and represents them as a simple linear function of time. It also goes briefly through the geometric data structures to solve the challenges eg Range Search of moving objects.

References

1) L. Forlizzi,R.H. Güting,E. Nardelli & M.Schneider.A Data Model and Data Structures for Moving Objects Databases. ACM SIGMOD Int.Conf.on Management of Data (SIGMOD), 319-330, 2000

2) [GDRS95] Guting, Ridder and M. Schneider. Implementation of the ROSE Algebra : Efficient Algorithms for Realm-Based Spatial Data Types.

3) [EGSV98] Erwig, Guting, M.Schneider and Vazirgiannis. Abstract and Discrete Modeling of Spatio-Temporal Data Types.

4) [EGSV99] Erwig, Guting, Ridder, M.Schneider, Vazirgiannis. Spatio-Temporal Data Types: An Approach to modeling and Querying Moving Objects in Databases.

5) [GS95] Guting and M.Schneider. Realm Based spatial Data Types : The ROSE algebra

6) [GBE+98] Guting, Bohlen, Erwig, Jensen, Lorentzos, M.Schneider, Vazirgiannis. A foundation for representing and querying moving objects

7) Spatial and Moving Object Databases: State of the Art and Future Research Challenges

8) Pankaj Agrawal, Lars Arge, Jeff Erickson, Hai Yu : Efficient Trade off schemes in Data Structures for querying Moving Objects

9) Data Structures for Moving Objects

10) Advances in Spatial and Temporal Databases : 7th International Symposium

11) Spatial Data Structures

12) An Introduction to Spatial Database System

13) Moving Object Databases : Modelling and Querying History of Movement

14) Adriano, Luca Forlizzi, Christian Jensen, V assilakopoulus : Access methods and Query processing Techniques

15) José Antonio Cotelo Lema, Luca Forlizzi, Ralf Hartmut Güting, Enrico Nardelli,Markus Schneider : Algorithm for Moving Object Databases

16) Advances in Spatial and Temporal Databases : 12th International Symposium

17) Computational Geometry : Algorithm and Applications 3rd Edition