Minimum Bounding Bicone: a new Minimum Bounding Box System for Trajectory Data

Liu Yinpei
lyp_bobi@yahoo.com
Shanghai Jiaotong University

Abstract

In the widely used Minimun Bounding Rectangle(MBR) system, if we want to express a moving object in a given time period, we have to form a 3-dimension rectangle that bound it, namely $[x_1, x_2][y_1, y_2][t_1, t_2]$. And when doing different kinds of query on the trajectories, we actually first use message of their MBRs to do some pruning to avoid the full scan of the database and the complex computation of the exact distance between trajectories. In this paper we propose a new method called Minimum Bounding Bicone (to make difference with Minimum Bounding Box, we abbreviate this as MBBC), which we would show to be a more precise description of the original trajectory points, which in turn would have stronger pruning power.

ACM Reference Format:
Liu Yinpei. 2019. Minimum Bounding Bicone: a new Minimum Bounding Box System for Trajectory Data. In . ACM, New York, NY, USA, 5 pages.

1 Basic Observations and Hypotheses

The index structure of spatial data is somehow matured now, as index structures like R-Tree, k-d Tree, space-filling curves, their variants, and other index structure have shown their efficiency on different realms in application. However, when comes to trajectory data, we find that these index structure can be further grained to become faster using the additional information and insight of real data. For trajectory data, we have some basic observations that is widely accepted:

1. Moving objects have only limited velocity, and the distance of a pair of adjacent points in one trajectory is hardly bounded according to the time difference of the points.

2. Enormous trajectory data is being produced every day, which naturally raised the necessity to use distributed databases and analysis tools.

, ,

3. The calculation of exact trajectory distance requires complex computation, so avoiding the exact calculation by pruning is very important. 4. The data are generally inserted in their time order, and updates to the past data is rarely needed.

So there are some basic hypotheses that we assume is true.

1. Most moving objects we are handling share a common acceptable speed bounds, like 30km/h for walking people, 200km/h for cars, 70km/h for ships. When handling spatio-temporal databases consists of two or more kinds of moving objects, build different indexes for different kinds is generally a good idea.

2. We assume the trajectory data is stored in a distributed environment like HDFS, which generally load a larger part of the data in a single I/O action. (For example, 64MB per block in HDFS instead of 8KB per page in Postgre SQL.)

3. We have to do calculation on the trajectories instead of the points in the trajectories, so the real computation cost of the queries is polynomially higher than queries on the points.

Studies on trajectories have proposed the inefficiency of Minimum Bounding Boxes for a long time [1], while good substitution to it are never provided to our knowledge. In this article, we would propose Minimum Bounding Bicone as a new bounding system for trajectory data.

2 Queries Supported

We would like to support two basic kind of trajectory queries, namely range query and kNN query. Both of the query have multiple variants and different variants may require computing costs with polynomial level difference, so we would first give our definition of the query.

Definition 1. *(ST-point) A Spatio-temporal point(abbreviate as ST-point) is a 3-tuple (x,y,t). The x and y stand for the location of the point, and the t stand for the time that the location is recorded.*

Definition 2. *(Trajectory) A trajectory is a list of ST-points sorted by their time value in ascendant order, namely*

$$[(x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n)],$$

$$t_1 \leqq t_2 \leqq t_3 \leqq ... \leqq t_n.$$

The trajectory is the sampled points of the real trajectory, which is in the form of the function

$$T^{real} : t \Rightarrow (x, y), t \in [t_1, t_n].$$

Notice that we assume we know exactly the starting and ending time of the real trajectory, namely t_1 and t_n.
For the points are generally sampled in a given rate in the real world, we could assume that the differences $t_2 - t_1, t_3 - t_2, ..., t_n - t_(n-1)$ are nearly equal, although we don't require this property in the rest of the paper.

Definition 3. *(Integral Distance) Given two trajectories $T_a = [(x_{a1}, y_{a1}, t_{a1}), (x_{a2}, y_{a2}, t_{a2}), ..., (x_{am}, y_{am}, t_{am})]$ and $T_b = (x_{b1}, x_{b1}, t_{b1}), (x_{b2}, y_{b2}, t_{b2}), ..., (x_{bn}, y_{bn}, t_{bn})$, we define their Integral Distance as below:*
Denote $\hat{t} = \hat{t_a} \bigcup \hat{t_b} = \{t_{a1}, t_{a2}, ..., t_{am}\} \bigcup \{t_{b1}, t_{b2}, ..., t_{bn}\}$, and $\check{t} = [t_1, t_2, ..., t_l]$ as the $\hat{t}$ sorted by its ascendant order. And for $t \in \hat{t}$ define

$$T'_a(t) = \begin{cases} T_a(t), t \in \hat{t_a} \\ T_a(t_{ak}) + (T_a(t_{a(k+1)}) - T_a(t_{ak})) * \frac{t - t_{ak}}{t_{a(k+1)} - t_{ak}}, \\ \quad t \in [t_{a1}, t_{am}], t_{ak} < t < t_{a(k+1)} \\ T_a(t_{a1}), t \in [t_{b1}, t_{bn}] \setminus [t_{a1}, t_{am}], t < t_{a1} \\ T_a(t_{am}), t \in [t_{b1}, t_{bn}] \setminus [t_{a1}, t_{am}], t < t_{am} \end{cases}$$

and $T'_b(t)$ similarly. Then the Integral Distance is

$$d(T_a, T_b) = \sum_{i \in \{1,2,...,l-1\}} \frac{1}{2}(T'_a(t_i) - T'_b(t_i) + T'_a(t_{i+1}) - T'_b(t_{i+1})) * (t_{i+1} - t_i).$$

This is a natural extension of the real Integral Distance

$$\int_{t_1}^{t_l} (T_a^{real} - T_b^{real}) dt.$$

Definition 4. *(Range Query)Given a query box*

$$[x_1, x_2] \times [y_1, y_2] \times [t_1, t_2],$$

return all the trajectories that "intersect", which means having at least one point inside, this box.

Definition 5. *(kNN Query)Given a trajectory which not necessarily contained in the database, and find the trajectories which have the k smallest Integral distance to the given trajectory in the database, which may contain itself.*

3 Caculation of MBBC

First we define a function which is at the core of this paper.

Definition 6. *(Possible Area) Denote C as the full spatial space, which is invariant over time. Given a spatial area $\mathcal{A}$(which could degenerate into a point) and a time stamp t_0, we define the function $Possible\_Area()$ which calculate the Possible_Area position at a given time t if the object could at most move at a velocity v.*

$$Possible\_Area(t; \mathcal{A}, t_0, v) = \{A + v|t - t_0| * \vec{n} \Big| ||\vec{n}|| = 1\}$$

Definition 7. *(Possible Cone) We further define the function to calculate all the Possible_Area points as Possible_Cone(), which could be further divide extend to $Possible\_Cone^+$ and $Possible\_Cone^-$ as tow nappes of the double cone.*

$$Possible\_Cone(\mathcal{A}, t_0, v) = \{Possible\_Area(t; \mathcal{A}, t_0, v) \times t\}$$

$$Possible\_Cone^+(\mathcal{A}, t_0, v) = \{Possible\_Area(t; \mathcal{A}, t_0, v) \times t \Big| t \geq t_0\}$$

$$Possible\_Cone^-(\mathcal{A}, t_0, v) = \{Possible\_Area(t; \mathcal{A}, t_0, v) \times t \Big| t \leq t_0\}$$

Definition 8. *(MBBC)Given a trajectory*

$$T = [(x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n)],$$

with the maximum speed v, the MBBC of it is

$$MBBC(T; v) = Possible\_Cone^+((x_1, y_1), t_1, v) \bigcap Possible\_Cone^-((x_n, y_n), t_n, v),$$

which is a bicone.

Similarly we define MBBC of two area laid at starting and ending time, expressed as $(\mathcal{A}, t_1), (\mathcal{B}, t_n)$ to be

$$MBBC(\mathcal{A}, \mathcal{B}; t_1, t_n, v) = Possible\_Cone^+(\mathcal{A}, t_1, v) \bigcap Possible\_Cone^-(\mathcal{B}, t_n, v),$$

which shape would be like two truncated cones, and we still call it MBBC.

Due to the strategy of building the index tree, the $\mathcal{A}$and$\mathcal{B}$ of each MBBC would be rectangles, which may degenerate into a line segment or a point.

The pseudo code of MBBC and MBR would be like this.

The shape of the MBBC of a trajectory would like Figure 1. Notice that the volume of a cone is 1/3 of the volume of the cylinder, which would obviously result in stronger pruning power.

4 Building Index on BCs

MBBC is a tight expression to the raw trajectories, but the use of the maximum velocity leads to some problems when we want to combine multiple MBBCs to construct a RTree-like structure, as the slope of different MBBCs are different. To overcome this problem, we would combine multiple MBBC as they all moving at the maximum speed at this time period.

4.1 Global Index

We choose to build a B^+-Tree or a Hash table on the time axis, and for each time period(which is a leaf of the B^+-Tree or a value of the Hash table), we would build a RTree-like index structure called R^2-Tree. The square means that it use the range informations both at the start time and at the end

CLASS Class Statement

```
Class  MBR
  x1:Double
  x2:Double
  y1:Double
  y2:Double
Class  MBR_3d
  x1:Double
  x2:Double
  y1:Double
  y2:Double
  t1:Double
  t2:Double
Class  MBBC
  A:MBR //start area
  B:MBR //end area
  mbr_enabled:Bool //have exact MBR info
  mbr:MBR //the MBR info
Class  pointMBBC extends MBBC
  function x
      return A.x1
  end function
  function y
      return A.y1
  end function
```

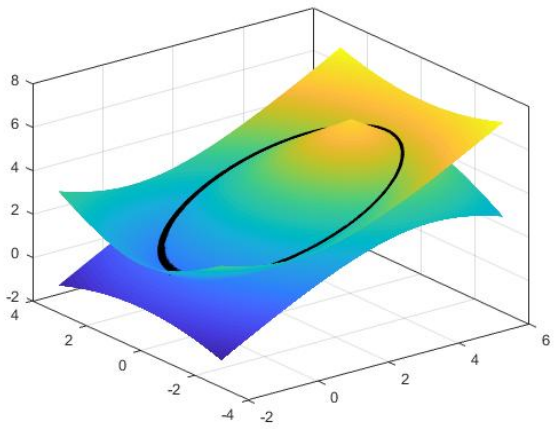

Figure 1. Plotted by MATLAB, with speed 1, start point (0,0,0), and end point (4,0,5)

time. We are doing this for four reasons. The first reason is the inefficiency of RTree and its variants on 3 or higher dimensions. This basically comes from the fact that the ratio of volume of the bounding boxes in comparison with the real volume grows exponentially with the increase of dimension.

The second reason is that the query on the trajectories generally only involves trajectories in a given time period instead of using time information as a measurement of the relation between trajectories. The third reason is that things can move forth and back in spatial axes, but can only move forth in time axes. Using B+-Tree or Hash table could utilize this knowledge better. And the last reason is that scholars have agreed on the necessity of cutting long trajectories into small segments to avoid their MBRs to grow too large. We would also use this method to avoid producing large MBBCs, and this method basically corresponds to the division of time axis.

We suggest two ways to build the first level of index upon time axis. The first one is to divide the time into segments of a given length, for example, one hour. Then we can access the R^2-Tree of the given time period by Hash Table. But this method may lead to data imbalance, for example, the car trajectories during the day is more than the trajectories during the night. The imbalance of data would not only lead to the imbalance of the whole tree, but also the imbalance of pruning power, that is, if there are a lot of trajectories in a given time period, we would want the time period to be smaller or the MBBCs would be overlapped with others frequently and make it hard to pruning using spatial information. So we suggest a second way, which is to build a B^+-Tree where each leaf node shares almost the same number of trajectory points. For general data, this method would result in longer querying time, but for skewed data, this method may produce better result.

In each time period, if the time period is $[t_1, t_n]$, we would build the so called R^2-Tree, which by structure is a binary tree, in a simple divide and conquer manner. At start, we have a root node which record all the trajectories in

$$MBBC(C, C; t_1, t_n, v).$$

Then we would either do a "start division" or a "end division", the process would be similar to the construction of k-d Tree.

For start division, we pick an area $\mathcal{A}$ and calculate the

$$\mathcal{B} = MBR(Possible\_Area(t; \mathcal{A}, t_1, v)).$$

Then we calculate

$$\mathcal{B}' = MBR(Possible\_Area(t; C \setminus \mathcal{A}, t_1, v))$$

which is the area that an object can reach if it start in $C \setminus \mathcal{A}$. So the left node of the root is the trajectories in

$$MBBC(\mathcal{A}, \mathcal{B}; t_1, t_n, v),$$

and right node is the trajectories in

$$MBBC(C \setminus \mathcal{A}, \mathcal{B}'; t_1, t_n, v).$$

The end division is just the same except we first pick an area containing ending points, and calculate back the Possible_Area starting position.

The area selection is just like the k-d Tree, where we first select an axis, and for the axis, we select the median point and divide the plain into two parts which would divide the data into two sets containing equal number of trajectories.

For there are two spatial dimensions, we have four division strategies, namely start/end division along x/y axis.

Due to the division strategy, the starting and ending area of MBBCs would always be rectangle, which we mentioned before. This makes calculation of checking intersection easier.

In every step, we would choose the one from the four strategies whose child nodes contains least volume. So although the MBBCs are overlapping each other inevitably, we could still try to avoid the MBBCs to overlap with a query box, which intuitively would lead to better pruning result. If two or more of the strategies have the same child nodes volume, we would randomly pick one.

The division procedure would end when each partition is small enough, which generally should be decide by the platform, for example, 64MB or 128MB for HDFS. If data update is needed, we would suggest to use only 40% to 80% of the maximum capacity of a leaf node in case their would be more data to insert. As stated earlier, we suppose that we don't have to change the past data a lot, so this structure would generally be good enough.

The leaf node of the R^2-Tree would contain these messages: full start rectangle, full end rectangle, actual start rectangle, actual end rectangle, actual maximum speed, and the reference to the MBBCs of the contained trajectories. the difference between full start rectangle and the actual start rectangle is that the actual start rectangle is the actual MBR of the starting points, while the start rectangle records the area that R^2-Tree allocated to this leaf. The difference is that if we want to decide whether a trajectory belongs to this leaf, we use actual start rectangle to prune this leaf, while if we want to insert data, we use the full start rectangle to decide which leaf it belongs to, so when inserting data, we don't have to create new leaf nodes.

4.2 Local Index

A shortage of the R^2-Tree is that the real velocity of the trajectories is transparent in the internal nodes, as we just use the maximum velocity for the internal nodes. But as the modern distributed system tend to do I/O action in bulk mode, in other word, read a big bulk of data, like 64MB in HDFS, for one I/O action, instead of the classical mode of reading a page of 8KB or something like that each time. So now we don't have to consider the I/O cost at the leaf-side of the index tree too much. Instead, we just use the first level of index to find the related partition of the data, and load all these partition into memory for further processing. And the second liar of the index, which is referred as "local index" in multiple papers, was build in memory for further pruning and retrieval. This just basically allows us to overcome the shortage of MBBC method and utilize its full power.

The local index is just like the Global index, the only difference is that when use the maximum speed of this partition(which is recorded as "actual maximum speed" in global index) to build the R^2-Tree. At the leaf nodes, they don't directly contain the data, but still the reference to the data and the MBBC, so we could do the most specific pruning.

4.3 MBR Construction(Optional)

Although we use MBBCs to express the set of trajectories, we can still use MBR to enforce the pruning power. The MBR of each tree node, or each MBBC, could be calculated using a bottom-up methods, which simply combines the MBRs of it childs. But because we don't use MBR to build the tree, we have to calculate the exact MBR from every leaf node, which lead to read and analysis the whole data once more, thus lead to higher index-construction time.

5 Querying using Index

Intuitively, the calculation of deciding whether a query box intersects with a MBBC, and the calculation of calculating the minimum distance from a MBBC to a MBR or another MBBC seems to be time costing. That is what people generally argues about for an oriented bounding box system, which is widely discussed in the realm of collision detection.(They call the MBR along axes as AABB method, and not along axes as OBB method.) But these calculation of MBBC could be tolerable using the property of the bicone.

Consider first

For a query box $[x_1, x_2] \times [y_1, y_2] \times [t_1, t_2]$ in time period $[t_0, t_n]$, for the internal nodes, we first calculate

$$\mathcal{A} = MBR(Possible\_Area(t_0; [x1, x2] \times [y1, y2], t_2, v)$$

and

$$\mathcal{B} = MBR(Possible\_Area(t_n; [x_1, x_2] \times [y_1, y_2], t_1, v)$$

The calculation is done only once so although we have to do some more calculation, the cost would not be large. And this is the first level of pruning which may produce lot of false positive. Then we would just compare $\mathcal{A}$ and $\mathcal{B}$ with the MBBC's shape at start and end time.

And for a leaf node, which stand for a pointMBBC, due to the fact that the intersection of two cone's surface is an ellipse(it would be a combination of 8 line segments and 8 ellipse for a generall MBBC, and very complex to prune.)

Algorithm 2 Check if a MBBC of points intersects with a bounding box

```
1: function INTERSECT(a:pointMBBC, b:MBR_3d)
2:
3: end function
```

The two methods would both produce some false positive, but it's necessary in order to reduce the calculation cost.

6 Implementation on Spark

In this section we will show how to implement the two-level of global index and one level of local index in a widely used distributed platform, namely Apache Spark.

In Spark, there are naturally two levels of storage. The first level is RDD, which is open to the users, and stand for the data to be analysed. The second level is partition. Each RDD is divided into many partitions which is about 64MB each to store in HDFS, and when Spark have to analysis the data in a RDD, it just read the related partition to reduce I/O cost. The partition is generally transparent to users. The RDD is a structure optimized for analysis, so the insertion and deletion cost are very high.

Recent studies have proposed a widely used method called two-level index, which refers to a structure that use a global index to manage different partitions, and use a local index to handle the data inside a single partition.

Instead of just adopting the two-level index structure, we extend it into a three-level index using the property of time axis. This method would help us to append new data into the data set and the update of index more easily, which cater to the real application better.

At the top level, instead of represent the whole data as one RDD, we use a combination of RDDs to store data. Each RDD stands for a leaf node of the B^+-Tree or a hash value of Hash Map in the time axis of global index.

References

[1] Dieter Pfoser, Christian S Jensen, Yannis Theodoridis, et al. 2000. Novel approaches to the indexing of moving object trajectories.. In *VLDB*. 395–406.