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 Box(MBB) 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 $[x_1, x_2] \times [y_1, y_2] \times [t_1, t_2]$. And when doing different kinds of query on the trajectories, we actually first use message of their MBBs 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. And we designed a index structure named R^2-Tree, which could fully utilize the MBBCs to achieve better pruning power than R-Tree.

The MBBC and R^2-Tree method is actually a general method that can apply to many different existing trajectory systems to replace the MBB and R-tree. It have no special relationship between any definition of queries or any distance function/ dissimilarity measurement, and can be tuned to support many kinds of existing trajectory queries.

So in this paper we just implement MBBCs and R^2-Tree on the most common condition to illustrate its efficiency, leaving the combination of MBBCs with more specified and advanced to readers.

ACM Reference Format:
Liu Yinpei. 2019. Minimum Bounding Bicone: a new Minimum Bounding Box System for Trajectory Data. In . ACM, New York, NY, USA, 8 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. Moreover, the maximum velocity distribution often have locality.(caused by traffic jam, storm, etc.)

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 assumptions.

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. Although we don't assume the trajectory data to be stored in a distributed environment like HDFS, we would implement the algorithm in a distributed environment to achieve better scalability, 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.) Note that the implementation on a non-distributed environment lead to no reduction of the pruning power of the index. So in this case we don't have to attain the absolute balance of the tree, instead, we just have to ensure the tree is roughly balanced. So the balance after each insertion or deletion is out of our concern.

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. This system come from the basic idea that if we use the speed information, we could represent a trajectory of object moving in an nearly uniform rectilinear motion a lot better than a MBB. And it's obvious that the trajectory of objects in a short time period always tend to be similar to a uniform rectilinear motion.

2 Caculation of MBBC

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

Definition 1. *(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.*

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

Definition 2. *(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.*

$$Possible\_Cone(\mathcal{A}, t_0, v) = \{Possible\_Area(t; \mathcal{A}, t_0, v) \times t | t \in [-\infty, \infty]\}$$

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

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

Definition 3. *(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 Minimum Bounding Bicone 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. As we can see, the intervening points contributes nothing to the MBBC after the maximum speed is given, so we call this type of MBBC a MBBC of points.

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, so for convenience we still call it MBBC.

We call a MBBC with its $\mathcal{A}$ and $\mathcal{B}$ both rectangles, which may degenerate into a line segment or a point, as a regular MBBC. Due to the strategy of building the index tree, the $\mathcal{A}$and$\mathcal{B}$ of each MBBC mentioned in this paper would be rectangles. So in the rest of paper we assume every MBBC is regular without extra explanation.

Algorithm 1 Class Statement

```
Class Point
    dimension:Int
    x:Double
    y:Double
    t:Double
Class MBB
    dimension:Int
    low:Point //this Point stores the lower value of x,y and t for this MBB
    high:Point //this Point stores the higher value of x,y and t for this MBB
    function EXPAND(v:Double,t:Double)
        newmbb:MBB=this
        newmbb.low.t +=t
        newmbb.high.t +=t
        t=ABS(t)
        newmbb.low.x -= v*t
        newmbb.low.y -= v*t
        newmbb.high.x += v*t
        newmbb.high.y += v*t
        return newmbb
    end function
Class MBBC
    A:MBB //start area, dimension=2
    B:MBB //end area, dimension=2
    maxspeed:Double //the maxspeed of this layer
    mbb:MBB //the MBB when dimension=3
    function POSSIBLEA
        return B.expand(maxspeed, A.low.t-B.low.t)
    end function
    function POSSIBLEB
        return A.expand(maxspeed, B.low.t-A.low.t)
    end function
Class pointMBBC extends MBBC
    function x
        return A.x1
    end function
    function y
        return A.y1
    end function
```

The shape of a point 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.

A general regular MBBC would looks like Figure 2 in rough.

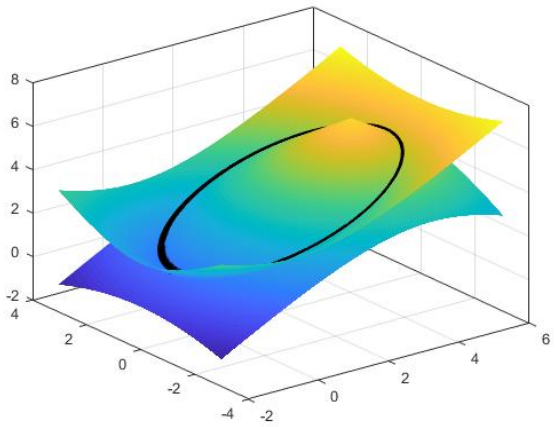

Figure 1. A MBBC of points,Plotted by MATLAB, with speed 1, start point (0,0,0), and end point (4,0,5)

Figure 2. A handmade hint to the shape of Regular MBBC. The edges should be replaced with cone surfaces.

3 Building Index on MBBCs

In most of studies on the similarity on the trajectories, they first state the queries and then design an index on that query. But in that we are designing a new structure to replace R-tree, instead of tuning R-tree to meet our need to specified types of query, we have to first state the index structure.

In fact, most of existing trajectory systems use some kind of R-tree structure. What we want is that they could directy apply their modification of MBB and R-tree to MBBC and R^2-tree to enhance pruning power.

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. Our solution is to build the tree structure using the maximum speed we already know, and after the tree construction, we recalculate the real bound of tree node from a bottom-up structure.

3.1 Temporal Index

As we stated earlier, dividing a long trajectories into multiple segments is necessary when indexing and querying trajectories. So the first level of index should divide the data into small time periods.

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 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 MBBs 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.

After time division, each trajectory would be divided into one or more segments. But the problem is some trajectories may not start at the starting time of the time period, and some trajectories may end before the ending time of the period, which made the data hard to use. So for convenience, for a trajectory segment that do not occupy the full time period, we record it's start point and end point, make a line through the points, and records the intersect points of this

line with the starting and ending time. And we use these two new points to make a MBBC. It's obvious that the new MBBC fully contains the old one.

3.2 Spatial Index

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

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

Then we calculate

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

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.

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 minimize the number of 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.

Due to the division strategy, the starting and ending area of MBBCs would always be rectangle. So all the MBBCs of the tree nodes is a regular MBBC, which we have mentioned before. This makes calculation of checking intersection easier.

The division strategy should stop when every node is small enough, which should roughly contain some data that can be read out by one I/O action (e.g. 8KB for Postgre SQL and 64MB for databases based on HDFS).

3.3 Exact MBB and max speed Construction

Although we use MBBCs to express the set of trajectories, we can still use MBB to enforce the pruning power. And we could label each MBBC with it's real max speed instead of the global one to further empower the pruning method. These two information of a tree node can be calculated directly by using the information of its child nodes. So by a bottom-up method applied to the tree after the division phase, we could append the information of exact MBB of the whole trajectory, the exact MBB of the starting area, the exact MBB of the ending area, and the exact max speed to each tree node to enhance the pruning power.

So every tree node of the index would contain two MBBCs, one for inserting new data so that every trajectory could be insert into a leaf node, the other for the exact message for the pruning.

Note that the MBBs and real maxspeed may have to update when new data is inserted or old data is modified.

3.4 Building a two level Index

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.

To build a global and local index doesn't have much difference to the construction of one level of index. The only difference is that the global index part contains the part of temporal index, while the part of local index only contains the R^2-Tree.

When building global index, the tree-node 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.

4 Calculations of MBBCs

The queries on the trajectories could be mainly divided into two categories. The first is queries related to an area, in which we have to retrieve trajectories that have some specified type of relation to the area. Range query and circle range query is the most basic member of this type. The second is queries on the similarity of the trajectories, where

we should use some specified trajectory distance functions or trajectory dissimilarity measures to get the querying results. KNN query and trajectory distance query belongs to this type. The more complicated queries, like Distance join, could be treat as the combination and extension of these two basic types.

To answer the two types of the queries, the most important questions we have to answer when using the index are as follow:

1. How to decide whether a MBBC intersects with a point, a MBB or another MBBC with low computation cost?

2. How to estimate the minimum and maximum distance between a MBBC and a point, a trajectory, or another MBBC with low computation cost?

3. How to prune the most irrelevant R^2-tree nodes with the minimum computation cost.

We believe MBBC could answer the first two questions better due to it is providing a tighter bounds than MBB. While for the third question, there are actually no optimization space, because MBB system only do several comparison action to prune a MBB. But we could do almost as good as it.

By theoretical analysis, MBBC with R^2-Tree would do better than MBB with R-Tree with area-related queries with rather "small" query box and perform not worse with the queries with relatively large query boxes(for these queries mainly only need to answer question 3). And they would do better on the distance related queries.

Intuitively, these calculations 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 MBB along axes as AABB method, and not along axes as OBB method.) But these calculation could be tolerable using the property of the regular MBBC.

4.1 Intersection detection

The intersection check of a point and a MBBC or two MBBCs in same time period is really easy with algorithms 2.

To answer the question 3, to any MBB or MBBC that serves as a query box, we first have to prune out most of really irrelevant data using most simple calculation, so we first calculate

$$\mathcal{P}_{\mathcal{A}} = MBB(Possible\_Area(t_0; [x1, x2]\times[y1, y2], t_2, maxspeed)$$

and

$$\mathcal{P}_{\mathcal{B}} = MBB(Possible\_Area(t_n; [x_1, x_2]\times[y_1, y_2], t_1, maxspeed).$$

Where the max speed is the theoretical one shared by this time period instead of the exact one. Using this, all the MBBCs that the starting area don't intersect with $\mathcal{P}_{\mathcal{A}}$ or ending area don't intersects with $\mathcal{P}_{\mathcal{B}}$ would be pruned easily. If we have the exact MBB message, we can instead use the message of MBBs to do this round of pruning by checking whether the MBB intersects with the query box.

Algorithm 2 MBBCIntersect

```
function Intersect(p:Point,bc:MBBC) p1:Point = the nearest verticle of bc.A to p p2:Point = the nearest verticle of bc.B to p
    if SpatialDistSqaure(p,p1)> bc.maxspeed^2*TemporalDistSqaure(p,p1)
    or SpatialDistSqaure(p,p2)> bc.maxspeed^2*TemporalDistSqaure(p,p2) then
        return false //use square distance to avoid using sqrt()
    end if
    return true
end function
function Intersect(bc1:MBBC,bc2:MBBC)
    if not Intersect(bc1.possibleB(),bc2.B) or not Intersect(bc1.possibleA(),bc2.A) then
        return false
    end if
    return true
end function
```

After the first level of pruning, we can use some more costy algorithm for the most of data is already pruned out. So we would start to check whether the MBBC actually intersects with the MBB.

A good property of bicone is that when looking from any direction which is perpendicular to t-axis, the shape of the bicone is a parallelogram with the same slope (which is the value of maximum speed v). We could easily proof that if a MBBC intersects with a MBB, then their projections on x-t plane, y-t plane and x-y plane also intersect. We would use this property for the second stage of pruning, named Projection Pruning.

Using the fact that the absolute value of the slope of each edge of the parallelogram is the same, we could transform the condition stated above into some simple comparison, as shown in Algorithm 4. Although there could be more pruning, we should consider that the exact computation of collision is unacceptable, and some false positive is acceptable.

The general version which handle MBBC instead of pointMBBC is similar.(incomplete here.)

4.2 Distance estimation

The mostly used Distance estimation is MINDIST, MAXDIST and MINMAXDIST. MINDIST and MAXDIST stand for the lower and upper bound of the two trajectory or bounding box, and MINMAXDIST stand for the minimum distance that is guranteed to find a trajectory in the bounding box. Other distance estimation may also use the MINDIST, MAXDIST

Algorithm 3 ProjectionPruning

```
function ProjectionPruning(bc:pointMBBC, box:MBB)
    maxspeed:Double = bc.maxspeed
    t0:Double = box.low.t-bc.A.t
    t1:Double = box.high.t-bc.A.t
    t:Double = bc.B.t-bc.A.t
    d:Double = maxspeed*t
    a:Double
    b:Double
    //check the y-t surface projection
    if bc.A.x+bc.B.x>box.high.x+box.low.x then
        a=bc.A.x-box.high.x
        b=bc.B.x-box.high.x
    else
        a=box.low.x-bc.A.x
        b=box.low.x-bc.B.x
    end if
    if a+b>d or a*t>d*t1 or b*t>d*(t-t0) then
        return false //pruning
    end if
    //check the x-t surface projection
    if bc.A.y+bc.B.y>box.high.y+box.low.y then
        a=bc.A.y-box.high.y
        b=bc.B.y-box.high.y
    else
        a=box.low.y-bc.A.y
        b=box.low.y-bc.B.y
    end if
    if a+b>d or a*t>d*t1 or b*t>d*(t-t0) then
        return false //pruning
    end if
    //check the x-y surface projection
    if d^2+(box.A.x-box.B.x)^2+(box.A.y-box.B.y)^2 > (2*box.x-box.A.x-box.B.x)^2+(2*box.y-box.A.y-box.B.y)^2 then
        continue
    end if
    if (bc.A.x-bc.B.x)(bc.A.y-bc.B.y)>=0 then
        if not InEllipse(bc,Point(box.low.x,box.high.y)) and not InEllipse(bc,Point(box.high.x,box.low.y)) then
            return false
        end if
    else
        if not InEllipse(bc,Point(box.low.x,box.low.y)) and not InEllipse(bc,Point(box.high.x,box.high.y)) then
            return false
        end if
    end if
    return true
end function
```

Algorithm 4 ProjectionPruning

```
function InEllipse(bc:pointMBBC,point:Point)
    if (bc.A.x-point.x)^2+(bc.A.y-point.y)^2+(bc.B.x-point.x)^2 + (bc.B.y - point.y)^2 > d^2/4 + (bc.A.x - bc.B.x)^2 + (bc.A.y - bc.B.y)^2 then
        return false
    end if
end function
```

and MINMAXDIST of the point pairs. These distance estimation produced good result on pruning, like [2]'s work on kNN query.

However, for the trajectory similarity pruning, the MINMAXDIST from a trajectory to a bounding box would always degenerate to its MAXDIST, because we have no knowledge about the trajectory points inside the bounding box. And because of the same reason, the calculation on MINDIST is also not effective and can only yield a loose estimation.

By using the MBBC method, we could yield tighter bounds to both MINDIST and MAXDIST, with tolerable computation cost.

The strategy we use is rather simple to reduce the computation. we first calculate the distance of two MBBCs at the starting and ending points, and combine them with max speed to yield a most pessimistic and optimistic distance as MINDIST and MAXDIST. The pseudo code is in Algorithm 5.

To answer question 3, when we are given a distance d and a trajectory traj, we don't have to run the code of calculating MINDIST and MAXDIST on every node in the tree. Instead, we can only concern the nodes that distance of the starting time plus distance of the ending time is lesser or equal to $3 * maxspeed * t + 2 * d/t$, which could be search quickly through the R^2-Tree

4.3 Querying across multiple segment

We should say that the queries result is nothing more than the combination of the single result of each related time period. For example [4] use $Hausdroff$ and $Fréchet$ distance, so they use a maximum function to combine the result of each time period. And [3] build multiple levels of trees and add the distance calculated from each level, which is the summary of each time period.

5 Comparison with MBB and RTree

Compared with MBB, MBBC takes advantage of the information of direction(in other world, starting and ending area) and maximum velocity. And the bound on maximum speed makes the trajectories start in a given area and end in a given area to be bounded in a bounding box, which naturally lead to R^2-Tree system.

The boost to pruning would be obvious. Because every trajectory is bounded in a smaller box than its MBB(not

Algorithm 5 MINDIST and MAXDIST

```
function MINANDMAX(bc1:MBBC, bc2:MBBC)
    maxspeed:Double = max(bc1.maxspeed,bc2.maxspeed)
    min:Double
    max:Double
    d1:Double = SPATIALDISTSQUARE(bc1.A,bc2.A)
    d2:Double = SPATIALDISTSQUARE(bc1.B,bc2.B)
    m1:Double = SPATIALDISTSQUARE(bc1.A,bc1.B)
    m2:Double = SPATIALDISTSQUARE(bc2.A,bc2.B)
    if d1 < (2*maxspeed*t)^2 and d2 < (2*maxspeed*t)^2 then //then they could meet
        min = (d1*d1+d2*d2)/maxspeed/4
    else
        t = bc1.B.t-bc1.A.t
        min = (sqrt(d1)+sqrt(d2))*t-maxspeed*t*t/2-2*(d1+d2)/maxspeed
    end if
    max = (sqrt(d1)+sqrt(d2))*t + maxspeed*t*t/2+2*(d1+d2)/maxspeed
    return (min,max)
end function
function MINANDMAX(traj:List[Point], bc:pointMBBC)
    maxspeed:Double = bc.maxspeed
    ts:Double = traj[0].t
    te:Double = traj[-1].t //-1 means the last one
    d1 = sqrt((traj[0].x+bc.A.x)^2+(traj[0].y+bc.A.y)^2)
    d2 = sqrt((traj[-1].x + bc.B.x)^2 + (traj[-1].y + bc.B.y)^2)
    summin:Double = d1* (traj[1].t- traj[0].t) + d2* (traj[-1].t-traj[-2].t)
    summax:Double = d1* (traj[1].t- traj[0].t) + d2* (traj[-1].t-traj[-2].t)
    for i = 1,2,3,...len(traj)-2 do
        summin += min(d1+(traj[i].t-ts)*maxspeed, d2+(te-traj[i].t)*maxspeed,0)* (traj[i+1].t-traj[i-1].t)
        summax += min(d1+(traj[i].t+ts)*maxspeed, d2+(te+traj[i].t)*maxspeed)* (traj[i+1].t-traj[i-1].t)
    end for
    return (summin/2,summax/2)
end function
```

exactly with MBBC, but we can use the intersection of MBB and MBBC for pruning, which must be smaller than MBB). That lead to a tighter bound to almost everything of the trajectories. And those information is particularly useful in the bottom of the index tree, as the starting and ending areas tend to be smaller. For MBBC with large starting and ending areas, its volume would be similar to MBB, while for the ones with small starting and ending areas, it at most have 1/3 volume of MBBs.

Two kinds of questioning is always proposed to MBBC method. The first is that pruning on R^2-tree need more computation, and the second is that the intersection of multiple MBBC is too large and cannot yield good pruning result.

For the first questioning, in fact the pruning on the R^2-tree is just like the pruning on the R-tree, as we can transform a R-tree area directly to the corresponding projection area on the starting and ending plane on the R^2-tree with tolerable false positive. Moreover, we have shown in previous section that the second level of pruning won't ask too much calculation.

For the second questioning, in fact our MBBCs never intersects with each other on the $\mathbf{R}^2 \times \mathbf{R}^2 \times \mathbf{T}$ space instead of the $\mathbf{R}^2 \times \mathbf{T}$ space. And treat trajectory, which have time information, in this space is actually a more natural view than the old one.

This usage of information do have some side effects. The first one is that MBBC handle an object rotating in a small circle, especially when they rotating in a very high speed, badly. But experiences have told us that trajectories of objects often looks like a combination of line segments. And if there is a lot of this kind of special data, building an index separately would be a good solution.

6 Implementation

6.1 Queries Supported

In this section, we just defined the most natural and most common queries to avoid misunderstanding. We hope the usage on these most basic queries would reveal MBBC's power to you.

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 4. *(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 5. *(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 6. *(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*

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

This is a natural extension of the real Integral Distance

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

It's obvious that the Integral Distance is a natural extension of Euclidean Distance, and is the most natural distance between two trajectories with time information. We could also proof it is a metric, while most other trajectory similarity measures are not metric, like DTW, LCSS and EDR.

Definition 7. *(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 8. *(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.*

6.2 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, modified the past data, and update index more easily, which cater to the real application better.

(I doubt this is not a good method)

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 temporal index. So when modifying data, we don't have to reconstruct the whole data, instead, we could just reconstruct the related RDD.

In addition, since the data volume would be very large, we also designed a memory scheduling algorithm which take use of the spatial and temporal information, called spatial-LRU, to reduce the I/O cost.

This part is incomplete.

7 Further work

TODO:
1. Use MBB information to add some "if" statement in every algorithm.
2. Suppurt some point distance(like closest point to a given trajectory) calculation in distance estimation. OPTIONAL:
1. Support DTW distance
2. Support Spatial Join
3. Support Distance Query
4. Combine MBBC with Z-curve like structure instead of R-tree

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.

[2] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. 1995. Nearest neighbor queries. In *ACM sigmod record*, Vol. 24. ACM, 71–79.

[3] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. Dita: Distributed in-memory trajectory analytics. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 725–740.

[4] Dong Xie, Feifei Li, and Jeff M Phillips. 2017. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1478–1489.