UMD SWENSON COLLEGE OF SCIENCE & ENGINEERING

UNIVERSITY OF MINNESOTA DULUTH

Driven to Discover

Scalable Techniques for Trajectory Outlier Detection

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

Usman Gohar

Acknowledgements

- Dr. Eleazar Leal has always been available for technical help and patient with me
- Faculty and Friends at UMD for encouraging me

Outline

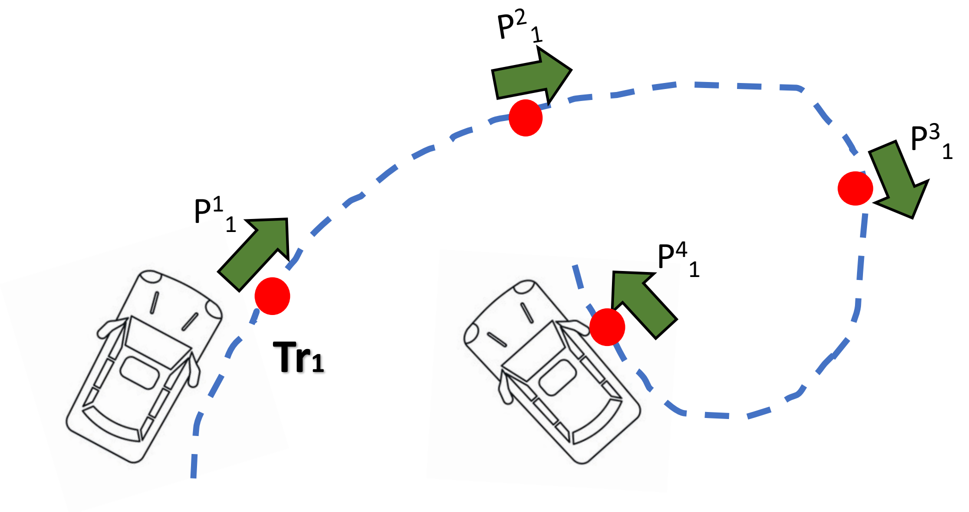
- Introduction
- Literature Review
- Methodology
- Results
- Conclusion & Future Work

What is a Trajectory?

- A single multidimensional point P^j_i generated from a moving object *O*, using a tracking device, at time-bin *tj*, is called a trajectory point of the trajectory Tri.

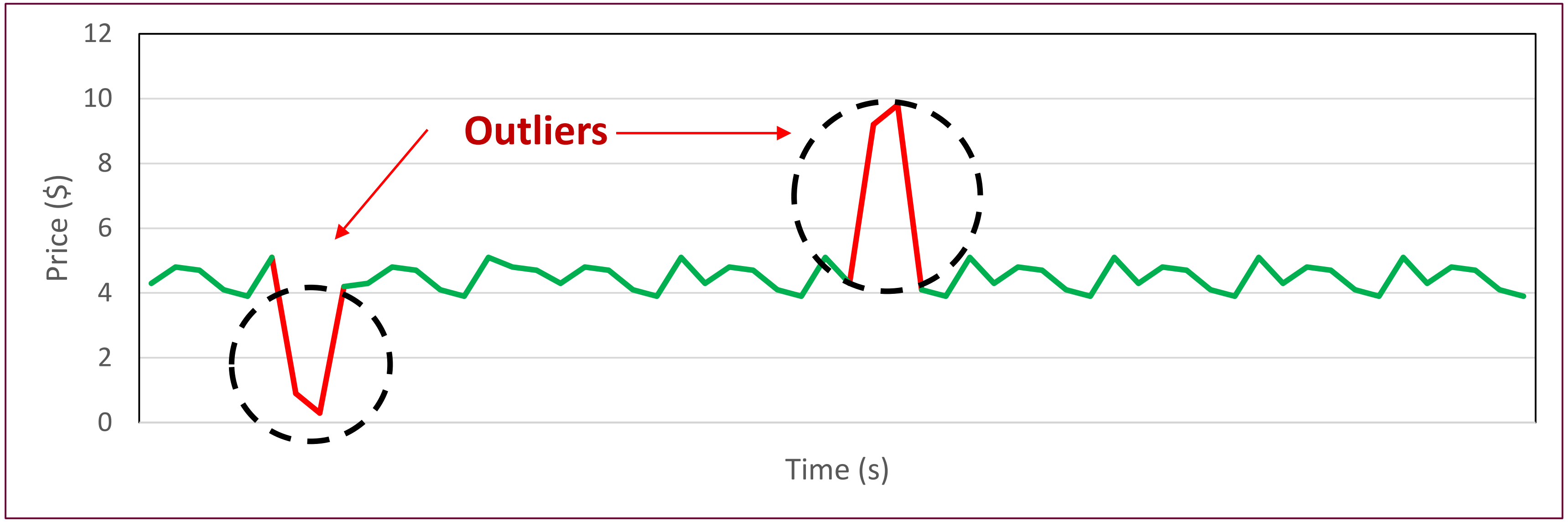
- The trajectory of a moving object *O* is then defined as a sequence of such trajectory points produced at time-bins $\{t1, t2 \ldots\ldots t_j\}$ denoted as $Tri = \{P^1_{i,} P^2_{i,} \ldots\ldots P^j_i\}$ as shown on the right

- A time-bin is the most smallest unit of time interval for a trajectory which can consist of a single or multiple geo-coordinate positions



What are Trajectory Outliers?

- Outliers are data points that behave inconsistently compared to the general pattern of data points in the all the trajectories in the window
- Example below shows two outliers in a stock price "trajectory"

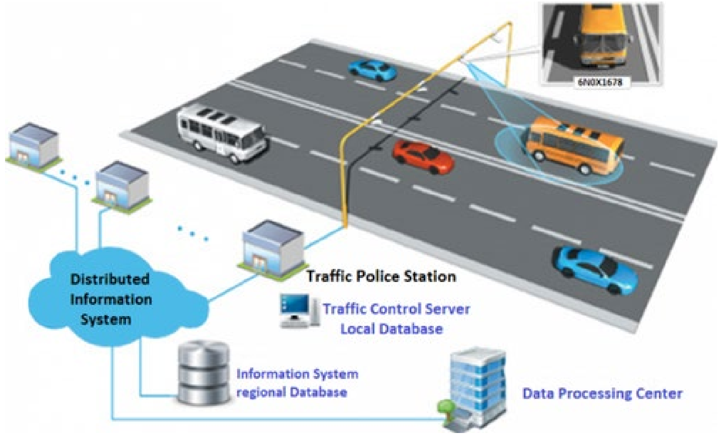


Applications

- Real time outlier detection is a time-critical process
- Applications include traffic management, stock-price monitoring

Source: Google Images

Imperative that outliers are detected in a timely fashion!!



Source: Google Images

Outline

- Introduction
- Literature Review
- Methodology
- Results
- Conclusion & Future Work

Literature Review

Outlier Detection in Static Datasets

- Distance Based Outliers (Knorr, 2000)
- ROAM (Li, 2007)
- iBAT: Detecting Anomalous Taxi Trajectories from GPS Traces (Zhang, 2011)
- A Taxi Driving Fraud Detection System (Ge, 2011)
- TRAOD (Lee, 2008)

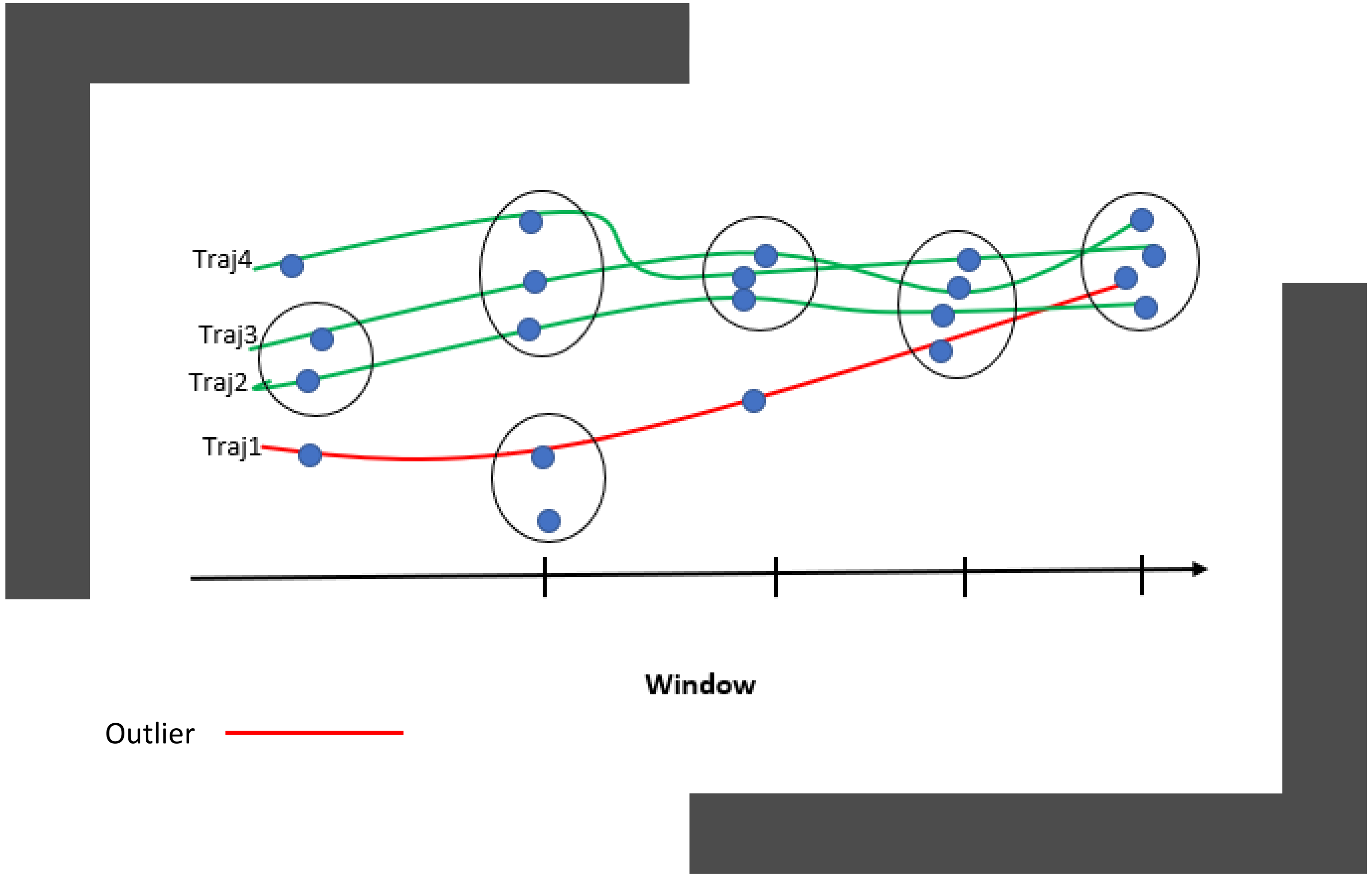
Outlier Detection in Data Streams

- Efficient Anomaly Monitoring over Moving Object Trajectory Streams (Bu, 2009)
- TOP-EYE (Ge, 2010)
- Outlier Detection over Massive-Scale Trajectory Streams (Yu, 2017)

Objectives

- The goals of this research are as follows:
  - Improve the execution time performance of the ODMTS algorithm through the use of spatial data structures
  - Propose a parallelization strategy for the ODMTS algorithm and study the scalability of this strategy in terms of execution time
  - Extend the TRAOD outlier detection algorithm to work in data streams

Outlier Detection over Massive-Scale Trajectory Streams



Parameters:

- Distance Threshold *d*
- Neighbor Count Threshold *k*
- Time-Bin Count *thr*

Definition:

Given a distance threshold *d*, neighbor count threshold *k* and time-bin count threshold *thr*, a trajectory is an outlier in the window if it has fewer than k trajectory neighbors i.e.

$$|TN(Tri,d,thrj)| < k$$

Haversine Approximation

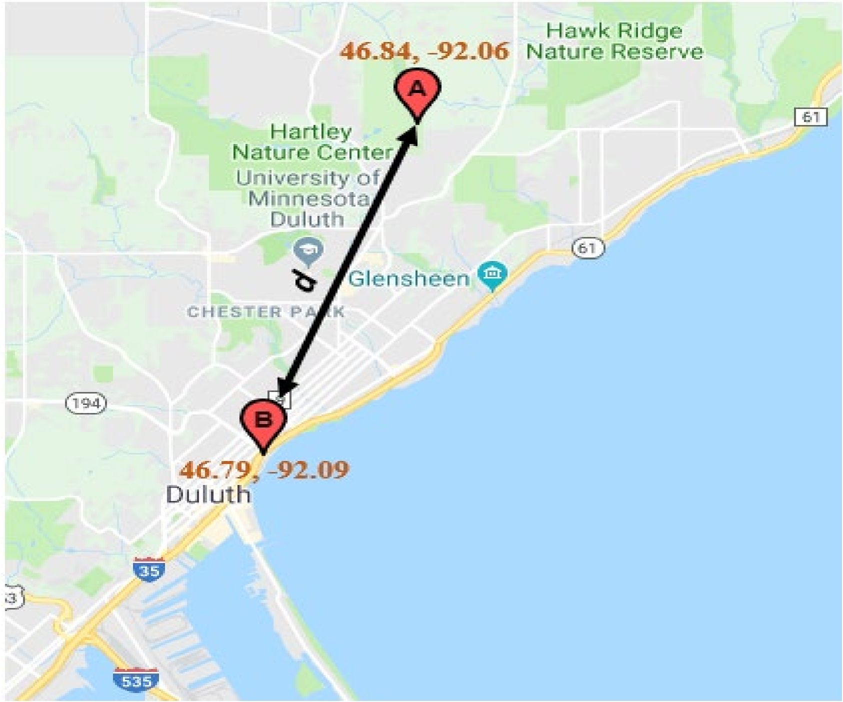
- Calculates the shortest distance between two points on a sphere given the latitude and the longitude

$$a = sin^2\left(\frac{\Delta\phi}{2}\right) + \ sin^2\left(\frac{\Delta\gamma}{2}\right) * \ \cos(\theta_1) * \cos(\phi_2)$$

$$c = 2 * \text{atan2}(\sqrt{a}, \sqrt{1-a})$$

$$d = R \ * c$$

where ϕ, γ, R is the latitude, longitude and the radius of the query search respectively



Data Structures of ODMTS

Parameters

thr
K
d

Tr_i. NT

Neighbors	Time-bins

Stores the information of all trajectories that share at least one point neighbor with Tr_i

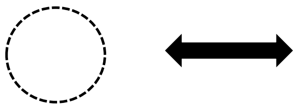
Tr_i. TList

Stores the IDs of neighboring timebins

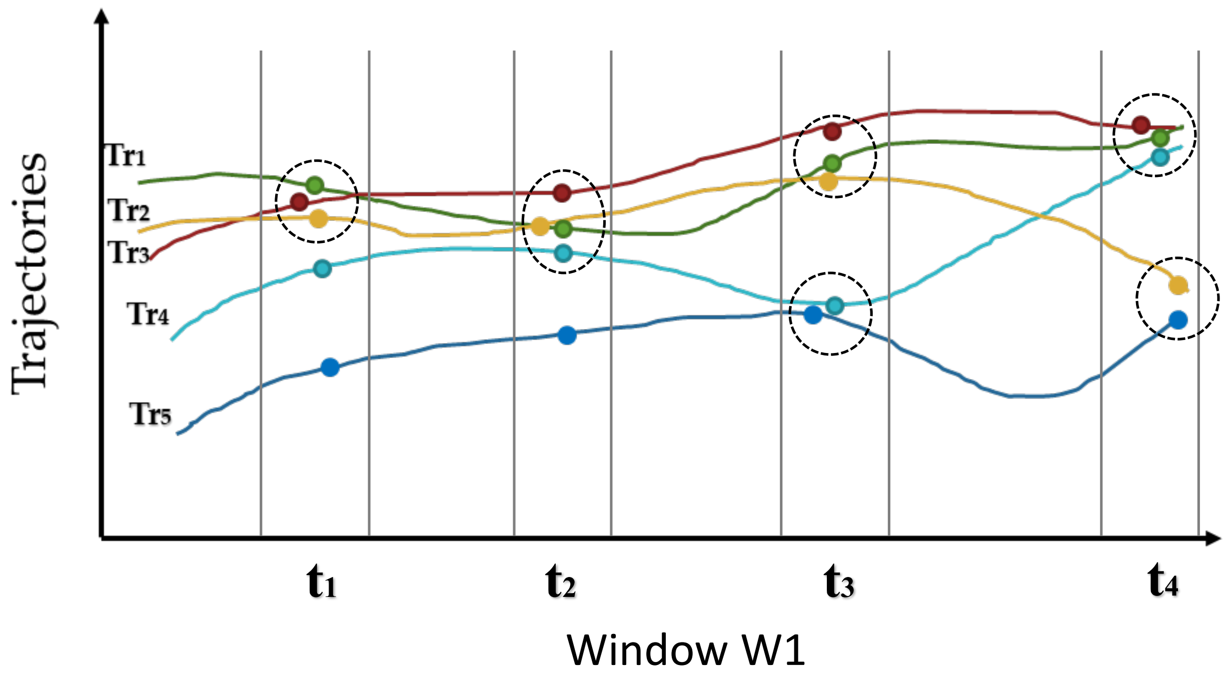
Range Query (Neighbors)

Find all neighbors of each point in each time-bin

D = 500m
Thr = 2
K = 3



Point Neighbor
($d < \varepsilon$)



ODMTS Example

Tr3. NT

Neighbors	Time-bins
Tr1	*t1*
Tr2	*t1*

|Time-bins| > thr ?

Tr3. Tlist

t1

Tr4. NT

Neighbors	Time-bins
Null	*Null*

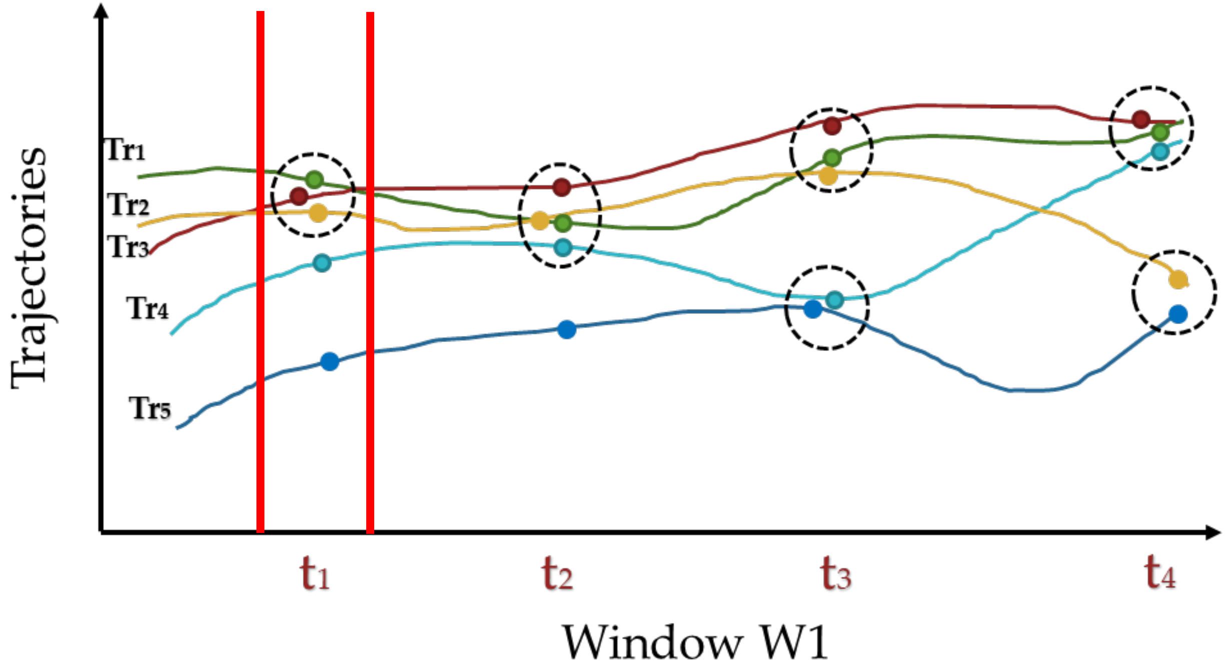
|Time-bins| > thr ?

Tr1. Tlist

Null

Time-bin t1

Time-bin t1



ODMTS Example

Tr3. NT

Neighbors	Time-bins
Tr1	*t1, t2*
Tr2	*t1, t2*
Tr4	*t2*

Tr3. Tlist

t1, t2

Tr4. NT

Neighbors	Time-bins
Tr1	*t2*
Tr2	*t2*
Tr3	*t2*

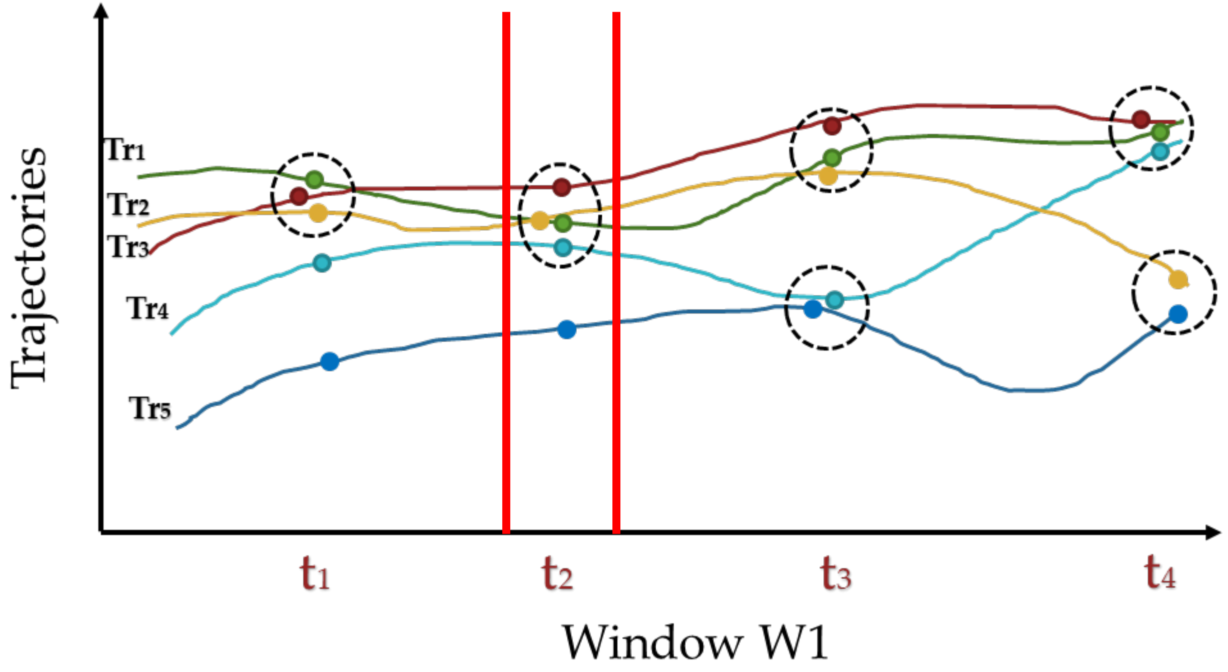
Tr4. Tlist

t2

Time-bin t2

t1, t2

Time-bin t2



ODMTS Example

Tr3. NT

Neighbors	Time-bins
Tr1	*t1, t2, t3*
Tr2	*t1, t2, t3*
Tr4	*t2*

|Time-bins| > thr ?

Tr3. Tlist

t1, t2, t3

Tr4. NT

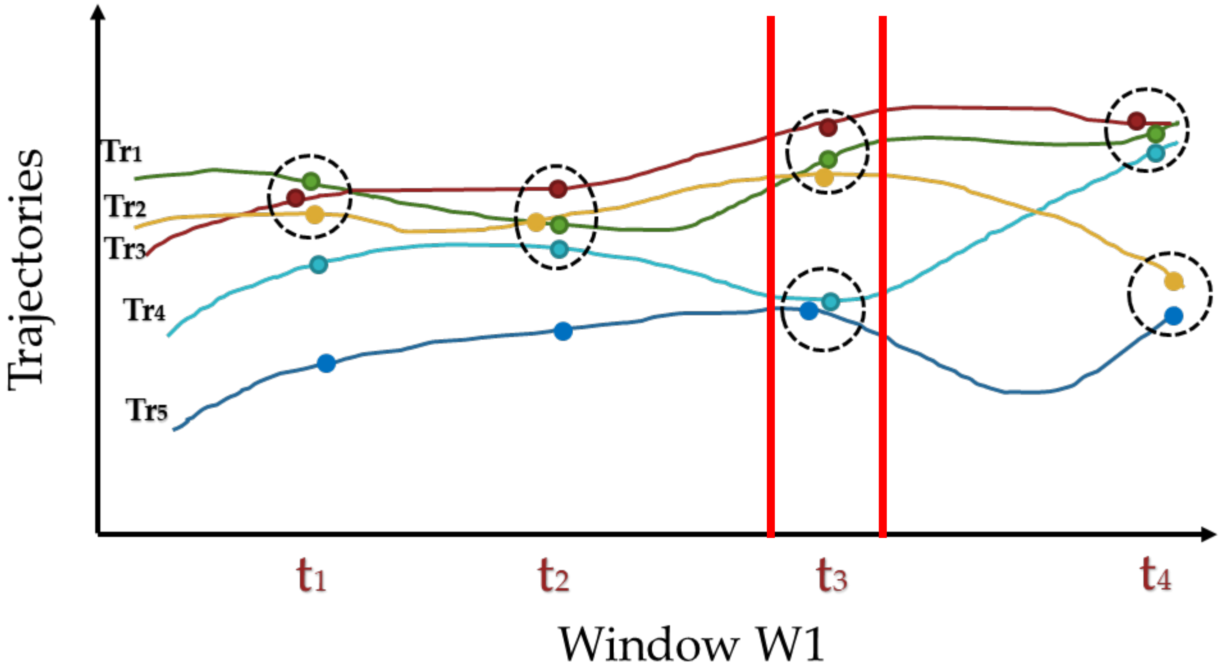
Neighbors	Time-bins
Tr1	*t2*
Tr2	*t2*
Tr3	*t2*
Tr5	*t3*

Tr4. Tlist

t2

Time-bin t3

Time-bin t3



ODMTS Example

Tr3. NT

Neighbors	Time-bins
Tr1	*t1, t2, t3, t4*
Tr2	*t1, t2, t3*
Tr4	*t2, t4*

|Time-bins| > thr ?

Tr3. Tlist

t1, t2, t3, t4

Tr4. NT

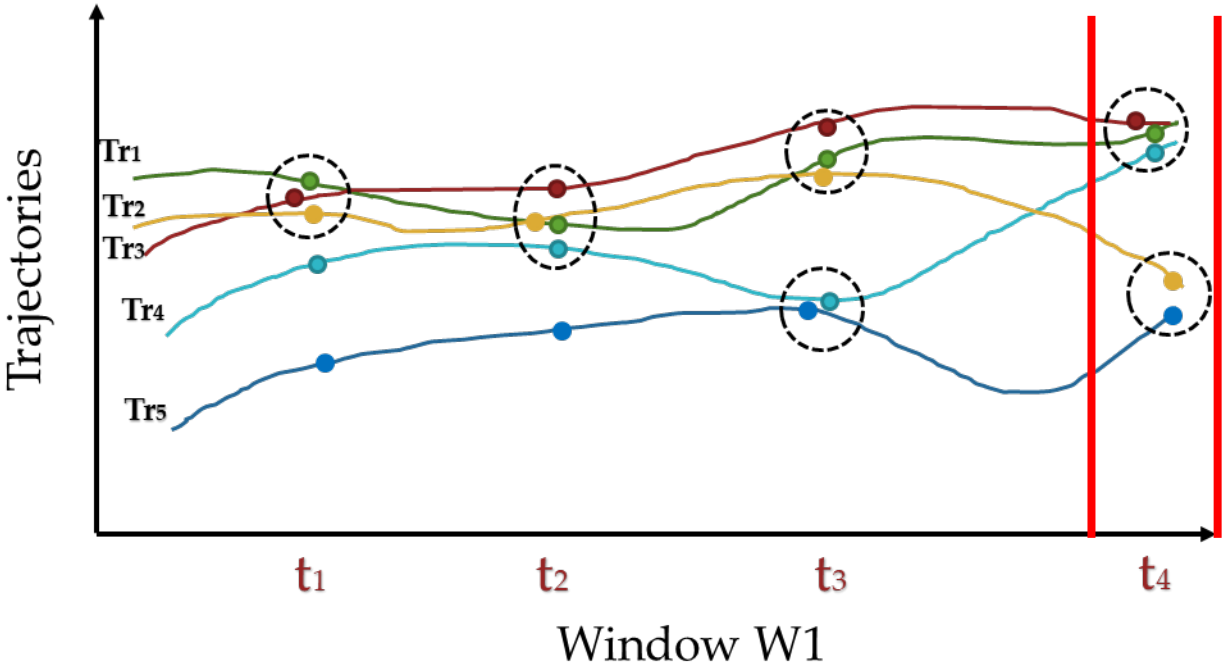
Neighbors	Time-bins
Tr1	*t2, t4*
Tr2	*t2*
Tr3	*t2, t4*
Tr5	*t3*

Tr4. Tlist

t2, t4

Time-bin t4

Time-bin t4



ODMTS Example

- Check each entry of Tr_i.NT:

  Are |time-bins| >= Thr_j ?

- If at least K entries fulfill the above condition, Trajectory is inlier.
- Else otherwise

- Hence, Tr_3 is an inlier and Tr_4 is an outlier

Outlier Test

Tr_3. NT

Neighbors	Time-bins
Tr_1	*t1, t2, t3, t4*
Tr_2	*t1, t2, t3*
Tr_4	*t2, t4*

Tr_4. NT

Neighbors	Time-bins
Tr_1	*t2, t4*
Tr_2	*t2*
Tr_3	*t2, t4*
Tr_5	*t3*

Tr_3. Tlist

t1, t2, t3, t4

Tr_4. Tlist

t2, t4

Time-bin t4

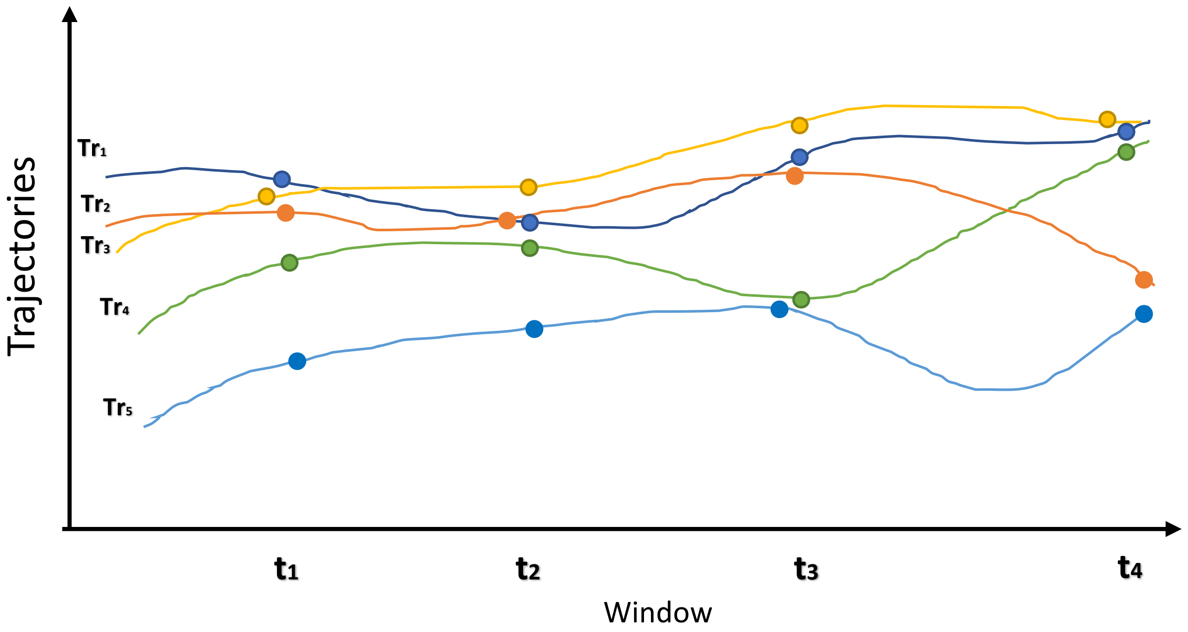
Pseudocode of ODMTS

Algorithm 1 Distance Based Outlier Detection (ODMTS)

Input Set of Trajectories, parameters: d, k, thr_i

Output Trajectory Outliers

for *each* TR_i **do**
    for *each* TR_k **do**
        if $dist(p_i^j, p_k^j) < d$ **then**
            $TR_i.NT.\text{insert}(TR_k)$
        end if
    end for
    if $TR_i.Count(t_{bin}) > thr$ **then**
        $TR_i.Tlist.\text{insert}(t_{bin})$
    end if
    if $TR_i.\text{size} < \text{k}$ **then**
        TR_i is an Outlier
    end if
end for



Outline

- Introduction
- Literature Review
- Methodology
- Results
- Conclusion & Future Work

Methodology

- We will now discuss our 3 approaches for improving the run-time performance of ODMTS

1. Using Spatial Data Structures to Speedup the Runtime
2. Parallelizing Strategy to Speedup the Runtime
3. Developing a Streaming Trajectory Outlier Detection Algorithm, PDMTS

Approach 1: Use Spatial Data Structures for Speed

Goal & Motivation:

- Use k-d trees and r-trees to improve running performance of ODMTS
- Compare the performances for both the trees
- Good for multi-dimensional points as in our case (latitude, longitude geo-coordinates)
- Our initial experiments revealed search query to be the slowest
- Suitable for range query search in large datasets

Key Idea Behind Approach 1

- Evaluated using python's cProfiler

Algorithm 1 Distance Based Outlier Detection (ODMTS)

Input Set of Trajectories, parameters: d, k, thr_i
Output Trajectory Outliers

for *each* TR_i **do**
  for *each* TR_k **do**
    if $dist(p_i^j, p_k^j) < d$ **then**
      $TR_i.NT.\text{insert}(TR_k)$
    end if
  end for
  if $TR_i.Count(t_{bin}) > thr$ **then**
    $TR_i.Tlist.\text{insert}(t_{bin})$
  end if
  if $TR_i.\text{size} < \text{k}$ **then**
    TR_i is an Outlier
  end if
end for

Pseudocode

- No change in pseudocode except insertion and search query
- Insert all geo-coordinates in tree
- Use a range query to retrieve neighbors in a radius *r*

Algorithm 1 Distance Based Outlier Detection (ODMTS)

Input Set of Trajectories, parameters: d, k, thr_i
Output Trajectory Outliers

for *each* TR_i **do**
    for *each* TR_k **do**
        if $dist(p_i^j, p_k^j) < d$ **then**
            $TR_i.NT.\text{insert}(TR_k)$
        end if
    end for
    if $TR_i.Count(t_{bin}) > thr$ **then**
        $TR_i.Tlist.\text{insert}(t_{bin})$
    end if
    if $TR_i.\text{size} < \text{k}$ **then**
        TR_i is an Outlier
    end if
end for

Experimental Setup

Hardware
- Akka
- 40 cores
- 512 GB RAM

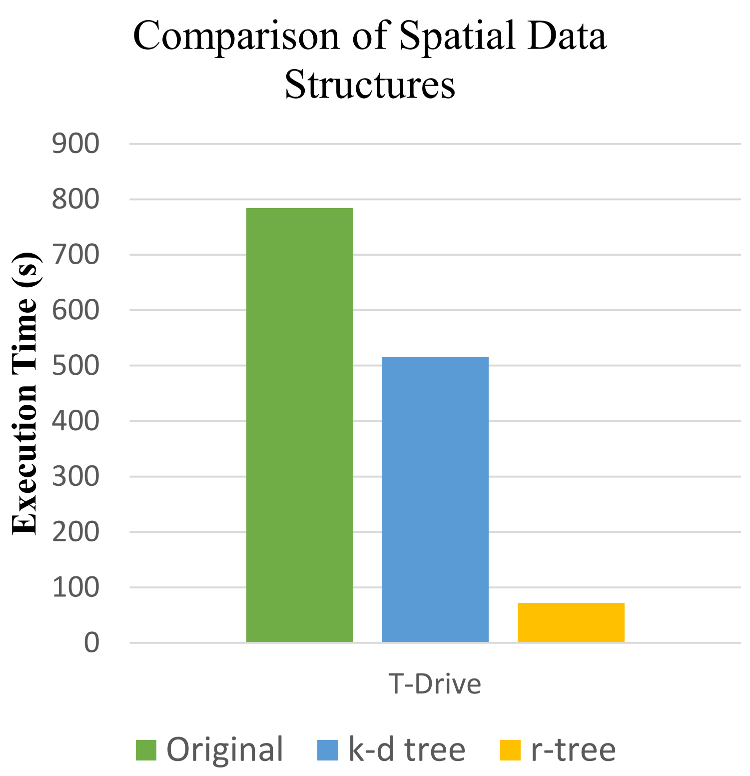
Performance Measures
- Total Execution Time

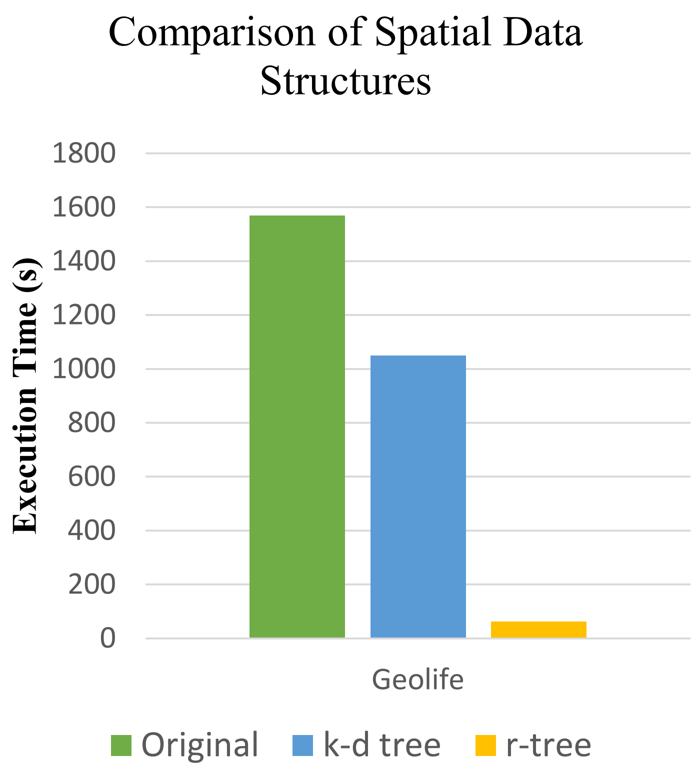
Software
- Python

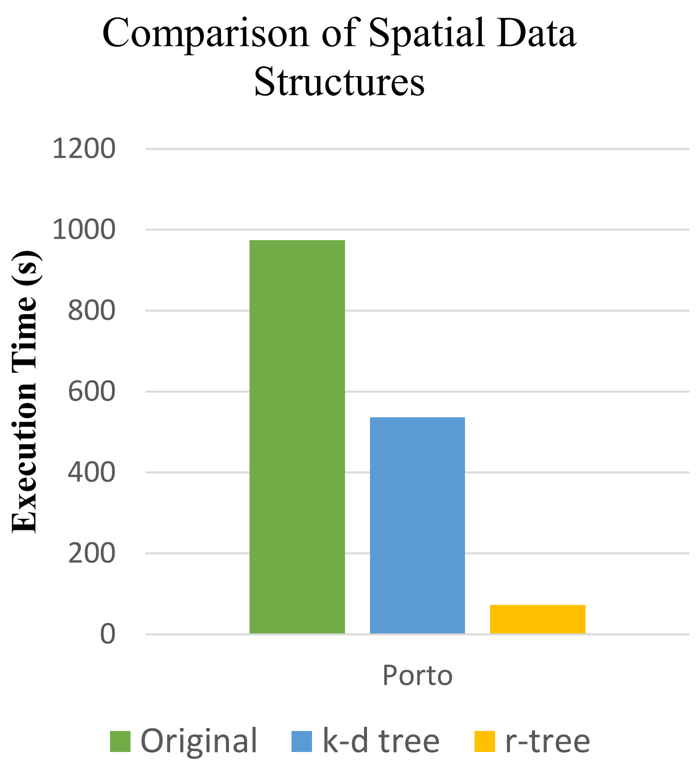
Datasets

Dataset	No. of Trajectories	No. of Points (million)	No. of Attr.	Duration	Distance	Area
T-Drive	10357	15	2	7 days	9 million	Beijing
Geolife	17621	23.6	7	5 years	1.2 million	Beijing
Porto	1710671	-	9	1 year	-	Porto

Results of Approach 1







The experiments showed that using an R-tree improved the execution time performance of the ODMTS algorithm by 10x!!

Approach 2: Parallelization Strategy

Goal & Motivation:

- Propose parallelization strategy for ODMTS
- Study scalability in terms of execution time
- Study memory usage vs performance gain

- Our initial experiments revealed search query to be the slowest
- Harness the computing power of modern machines

Pseudocode

- Divide the workload among cores
- Single tree, passed to each core
- Each core performs search query on a subset of the dataset

Algorithm 2 Parallel - Distance Based Outlier Detection (ODMTS)

Input Set of Trajectories, parameters: d, k, thr_i
Output Trajectory Outliers

Insert all trajectory points in K-D/R Tree
Divide the Trajectory Dataset into Count(cores)
Insert divided Datasets in separate TR_iList
for each core **do**
> Run a Ball-Point Query Search on all Trajectories in TR_iList
> Insert query search result in $Neighbor_iList$
> **for** each in $Neighbor_iList$ **do**
>> **if** $Count(Neighbor_iList[TR_i]) < k$ **then**
>>> TR_i is an Outlier
>>
>> **end if**
>
> **end for**

end for

Experimental Setup

Hardware

- Akka
- 40 cores
- 512 GB RAM

Performance Measures

- Total Execution Time

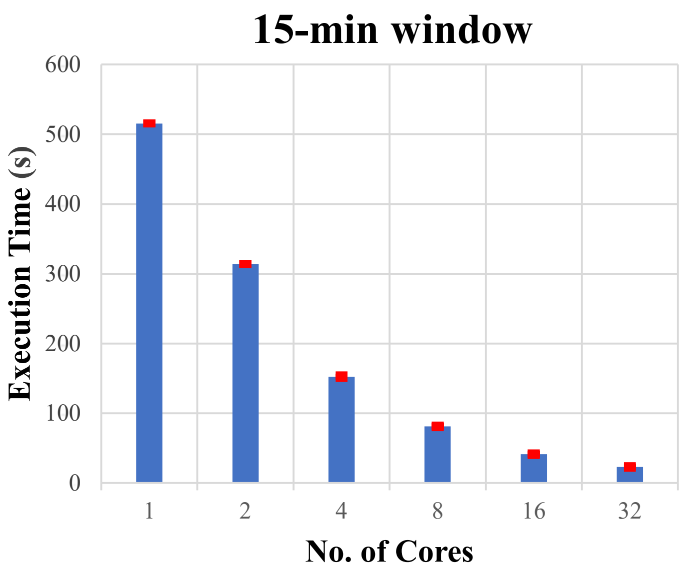
Software

- Python
- Multiprocessing

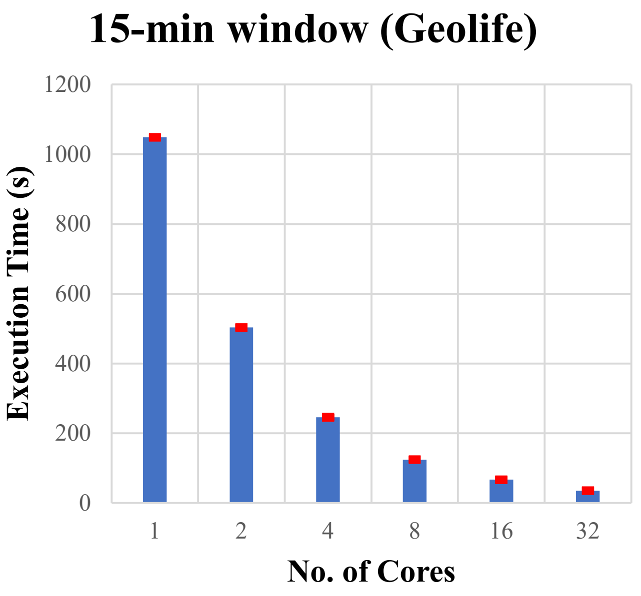
Datasets

Dataset	No. of Trajectories	No. of Points (million)	No. of Attr.	Duration	Distance	Area
T-Drive	10357	15	2	7 days	9 million	Beijing
Geolife	17621	23.6	7	5 years	1.2 million	Beijing
Porto	1710671	-	9	1 year	-	Porto

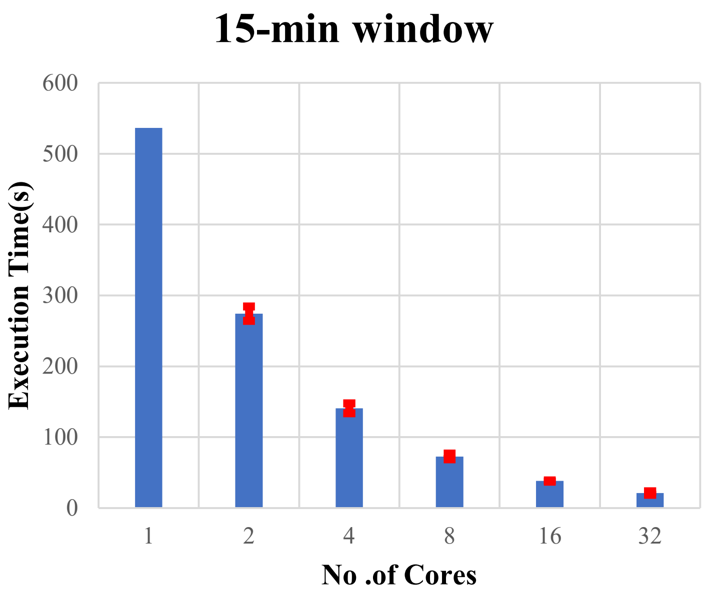
Results of Parallelization Strategy



T-Drive



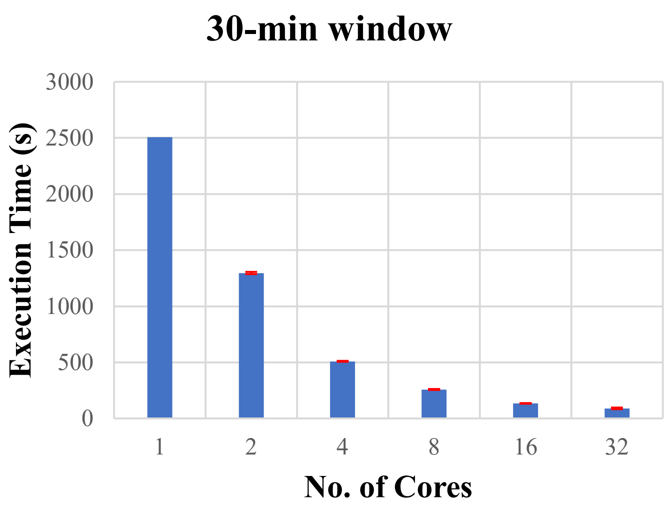
Geolife



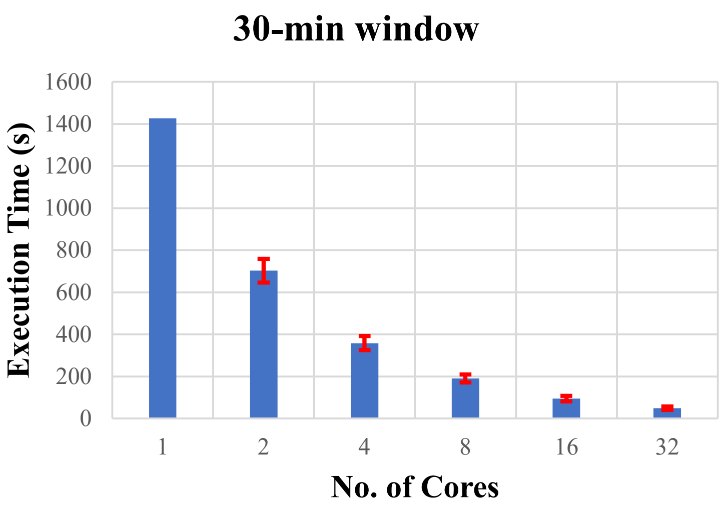
Porto

The experiments showed a linear decrease in execution time for k-d trees as the number of cores are increased!

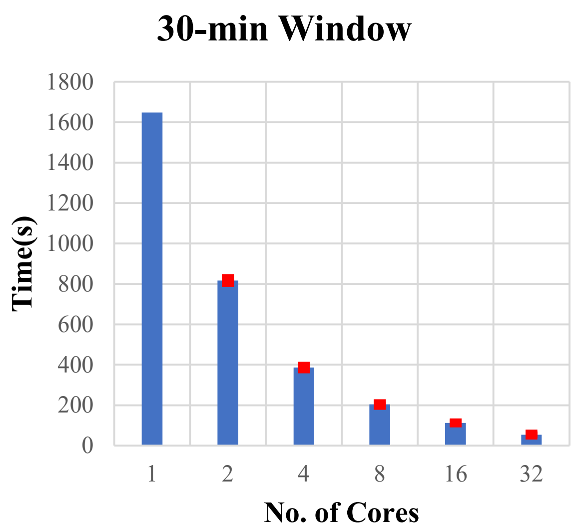
Results of Parallelization Strategy



T-Drive



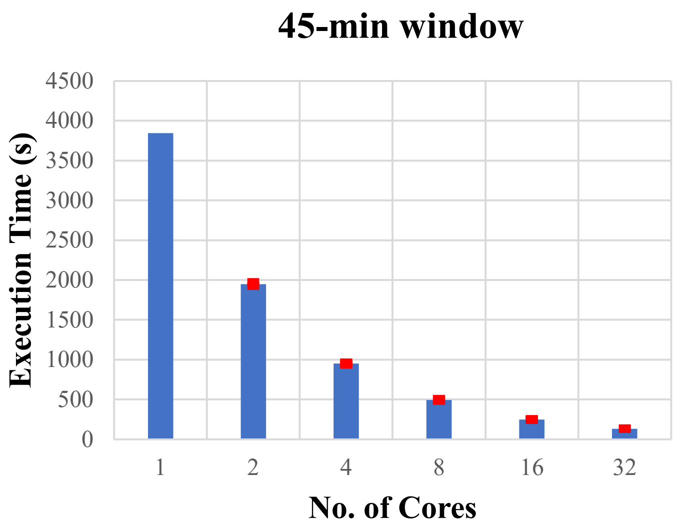
Geolife



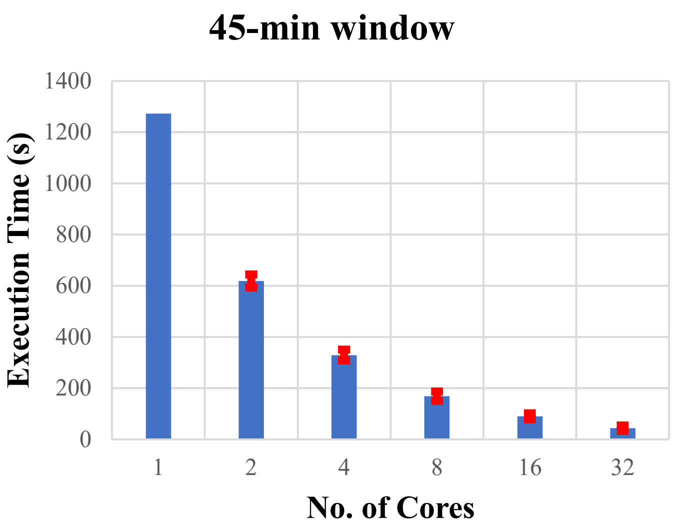
Porto

Workload is perfectly balanced among cores and the algorithm scales!!

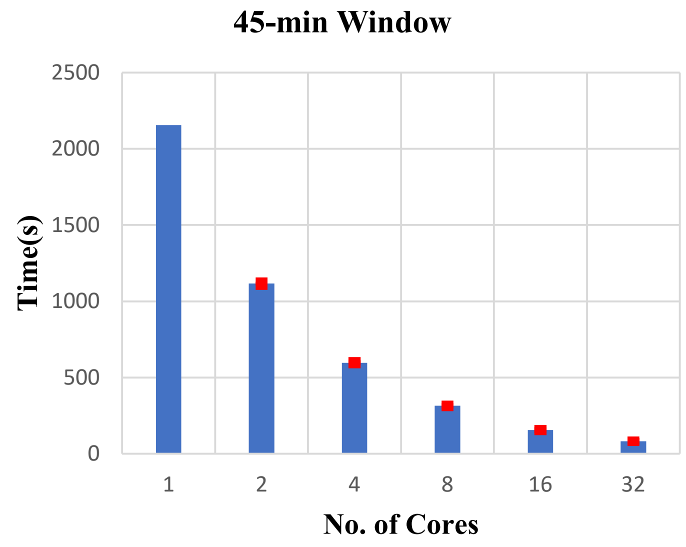
Results of Parallelization Strategy



T-Drive



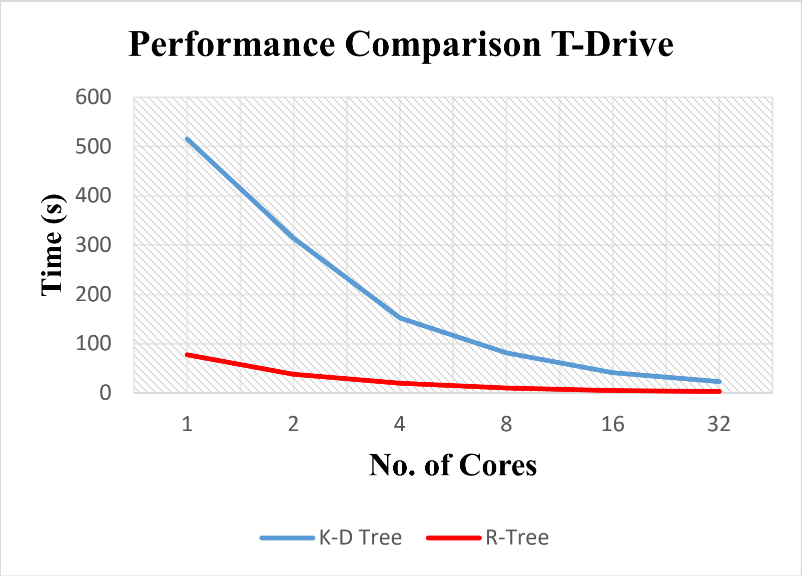
Geolife

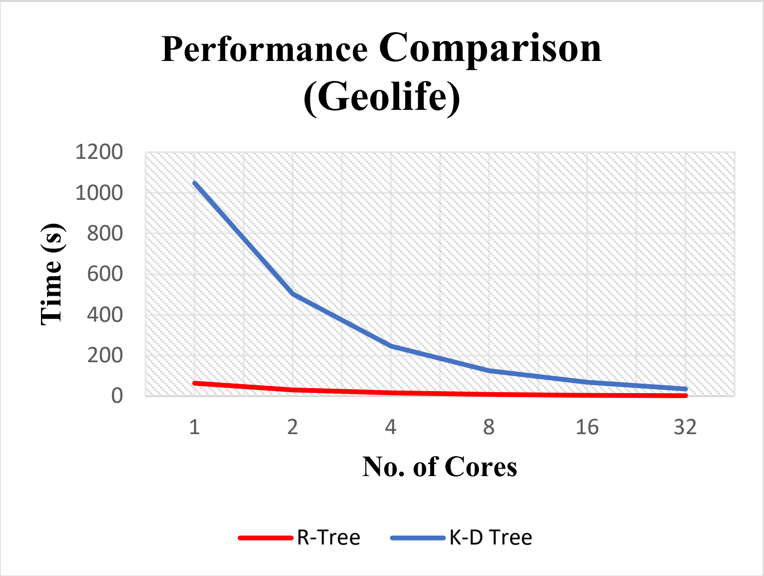


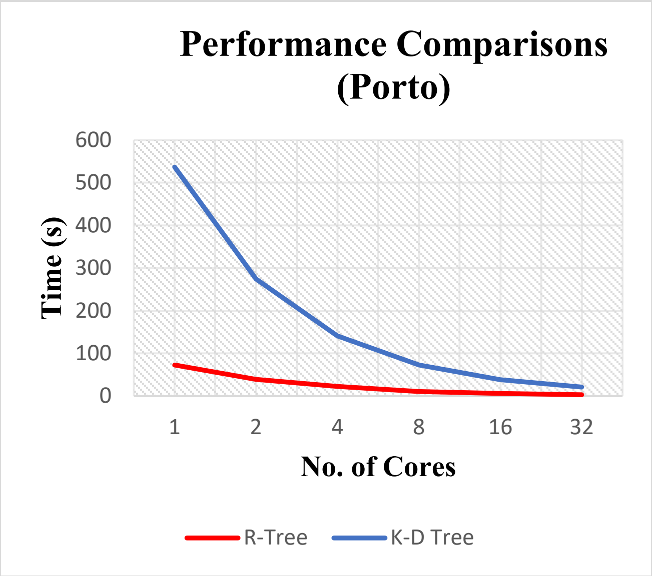
Porto

Scales linearly with the number of cores. In our case 32X decrease!

Results of Parallelization Strategy

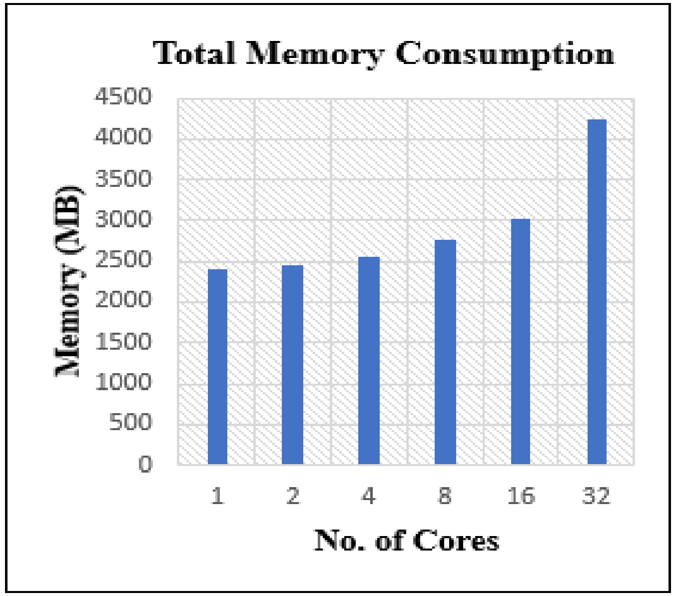




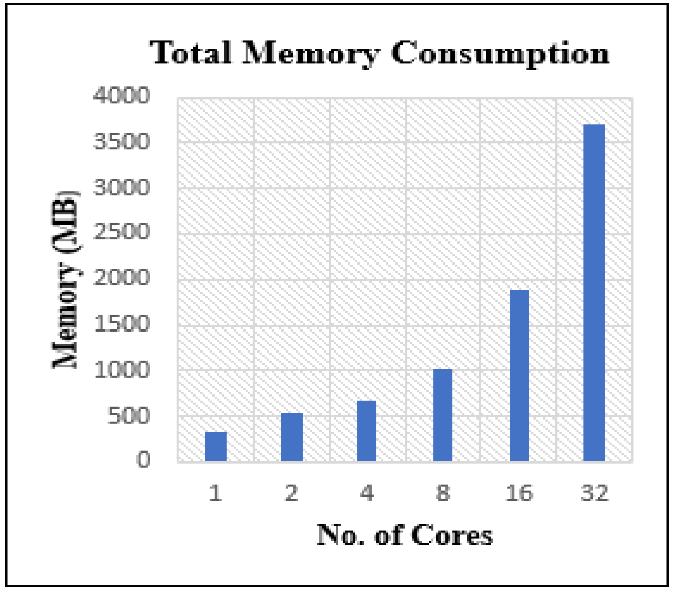


However, R-trees do not scale linearly in terms of execution times as the number of cores are increased, unlike k-d trees!!

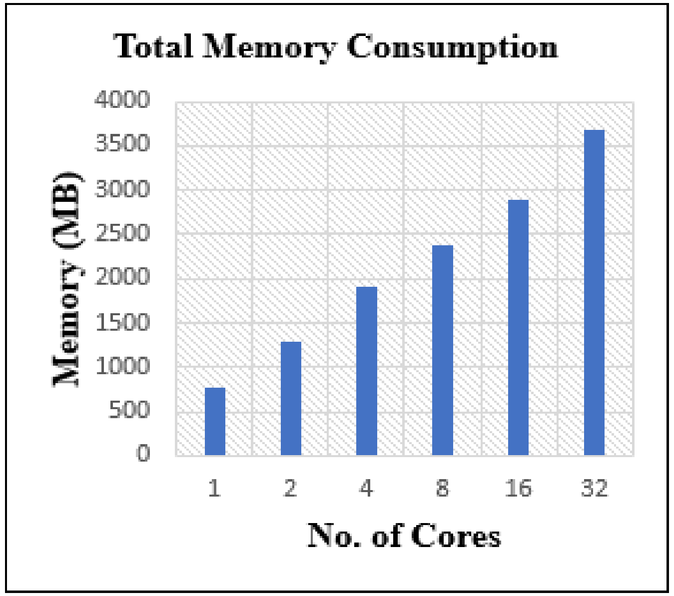
Results of Parallelization Strategy



T-Drive



Geolife



Porto

Total Memory Consumption shows an almost linear increase as number of cores are increased. This is because each core keeps a separate copy of the tree whilst having the same number of trajectory points!!

Approach 3: PDMTS

Goal & Motivation:

- Propose partition based streaming algorithm PDMTS
- Aim to detect trajectories that are significantly different but for a small period of the overall time of the trajectory
- Introduce windowing and temporal comparisons against other trajectories in the window
- Some outliers only hidden as sub-trajectories and not as a whole

Pseudocode

- Modifying TRAOD and Outlier Detection over Massive-Scale Trajectory Stream
- Detect outliers that are significantly different from the rest of the dataset but only for a small period of the trajectory
- Introduce windowing and temporal comparisons against other trajectories in the window
- First partition trajectory as in TRAOD then detect outliers using ODMTS
- Evaluate using Precision, Recall and F-Score

Algorithm 3 Streaming TRAOD (S-TRAOD)

Input Set of Trajectories, parameters: d, k, thr_i
Output Trajectory Outliers

—* Partioning Phase *—
for each TR_i **do**
  Partition TR_i at coarse granularity using MDL (L_i)
end for
—* Detection Phase *—
for each partition L_i **do**
  for each partition L_j **do**
    if $dist(p_i^j, p_k^j) < d$ **then**
      $TR_i.NT.\text{insert}(TR_k)$
    end if
  end for
  if $TR_i.Count(t_{bin}) > thr$ **then**
    $TR_i.Tlist.\text{insert}(t_{bin})$
  end if
  if $TR_i.\text{size} < \text{k}$ **then**
    TR_i is an Outlier
  end if
end for

Experimental Setup

Hardware
- Akka
- 40 cores
- 512 GB RAM

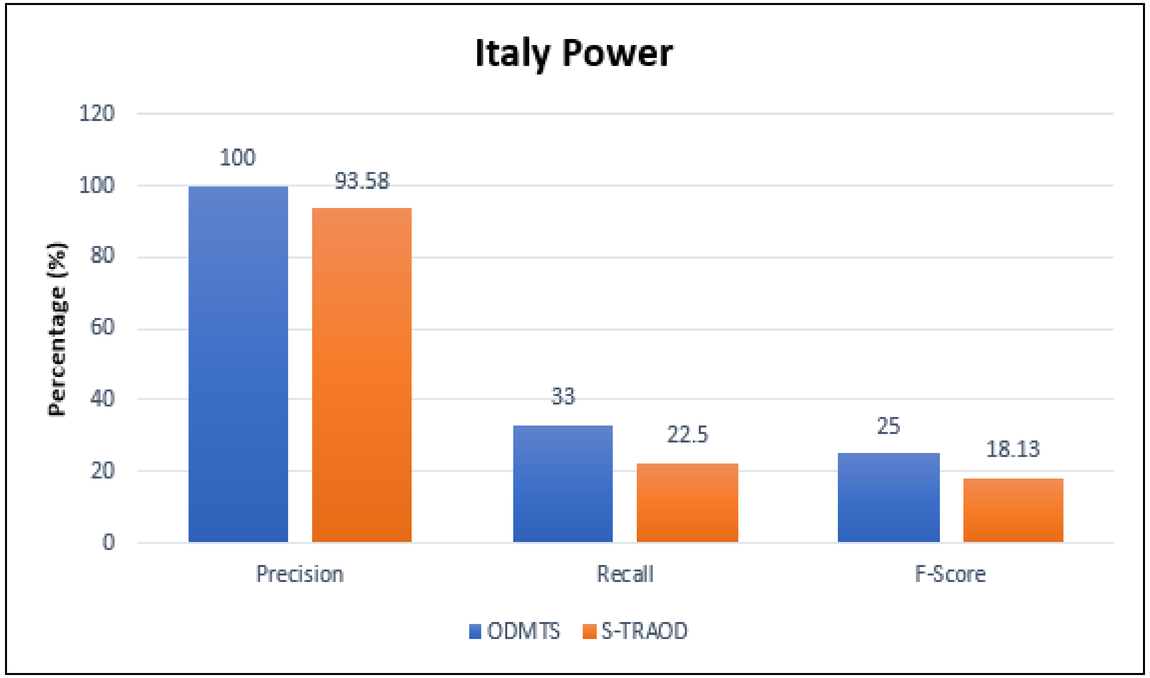
Performance Measures
- Precision
- Recall
- F-Score

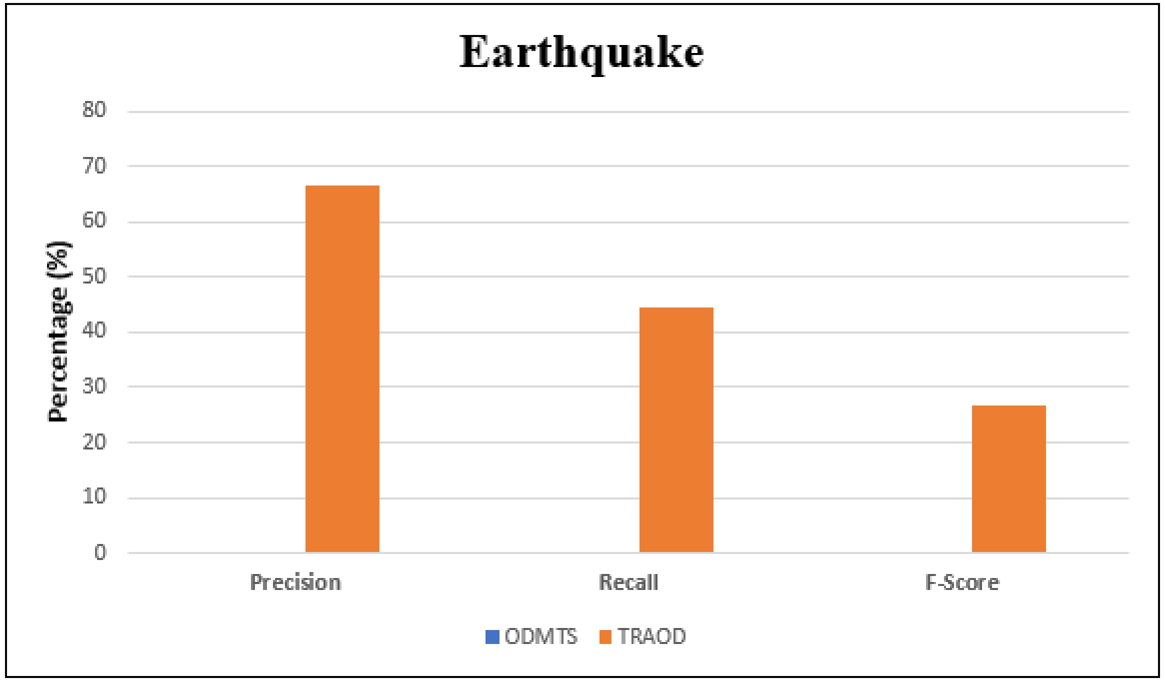
Software
- Python

Datasets

Dataset	No. of Trajectories	No. of Points (million)	No. of Attr.	Duration	Area
Earthquake	322	512	1	36 years	California
Italy Power	67	24	1	1 year	Italy

Results of Approach 3





ODMTS fails to detect any outlier on Earthquake dataset!!
Performs similarly on Italy Power!!

Outline

- Introduction
- Literature Review
- Methodology
- Results
- Conclusion & Future Work

Conclusions for Approach 1

- Slowest block of our pseudocode was the range query for the neighbor search.
- To address this, we used k-d trees and r-trees, which are spatial data structures, to improve range queries
- The experiments showed that using an R-tree improved the execution time performance of the ODMTS algorithm by 10x compared to without them

Conclusions for Approach 2

- We parallelize the ODMTS algorithm by dividing the workload across multiple CPU cores
- Linear decrease in the execution time of the ODMTS algorithm (32X)
- This suggests that the workload for the range query is equally balanced across all cores.
- However, the same was not observed with an R-tree.
- Execution time of the ODMTS algorithm is approximately 54 secs.
- The associated overhead cost of initiating multiple processes cancels out the performance gain
- Increasing the number of cores also showed an almost linear increase in memory usage.
- This was observed because each core keeps a separate copy of the tree and roughly all of them include the same number of trajectory points

Conclusions for Approach 3

- Aim to detect trajectory outliers that are significantly different from other trajectories but only for a very short period of the overall time of the trajectory.
- Real-life trajectory dataset outliers that exhibit this particular behavior are not detected by ODMTS.
- Our experiments showed that PDMTS detected almost 45% more outliers as compared to ODMTS for Earthquake dataset
- However, our experiments revealed that PDMTS was approximately 18% slower compared to ODMTS.
- Addition of the partitioning phase to the ODMTS algorithm

Future Work

- Explore further techniques to detect trajectory outliers that are not very significantly different than normal trajectories
- The challenge is to detect trajectory outliers that are in the guise of normal trajectories
- Evaluate the possible use of other distance measures such as Jaccard Similarity

References

- Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. 2000. Distance-based outliers: Algorithms and applications. The VLDB Journal 8, 3–4, 237–253
- Jae-Gil Lee, Jiawei Han, and Xiaolei Li. 2008. Trajectory outlier detection: A partition-and-detect framework. In Proceedings of ICDE. IEEE, 140–149
- Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. 2007. ROAM: Rule- and motif-based anomaly detection in massive moving object data sets. In Proceedings of SDM. SIAM, 273–284
- Wei Liu, Yu Zheng, and Sanjay Chawla. 2011. Discovering spatio-temporal causal interactions in traffic data streams. In Proceedings of SIGKDD. ACM, 1010–1018
- Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2013. T-Drive: Enhancing driving directions with taxi drivers' intelligence. Transactions on Knowledge and Data Engineering 25, 1, 220–232
- Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. IEEE Data Engineering Bulletin 33, 2, 32–40
- Yingyi Bu, Lei Chen, Ada Wai-Chee Fu, and Dawei Liu. 2009. Efficient anomaly monitoring over moving object trajectory streams. In Proceedings of SIGKDD. ACM, 159–168

QUESTIONS?