Big Mobility Data Analytics: Algorithms and Techniques for Efficient Trajectory Clustering

A thesis submitted in partial fulfillment of the requirements for the degree of *Doctor of Philosophy*

by

Panagiotis Tampakis

Department of Informatics
School of Information and Communication Technologies
University of Piraeus

Piraeus, 20 November 2019

Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Part I

Setting the Scene

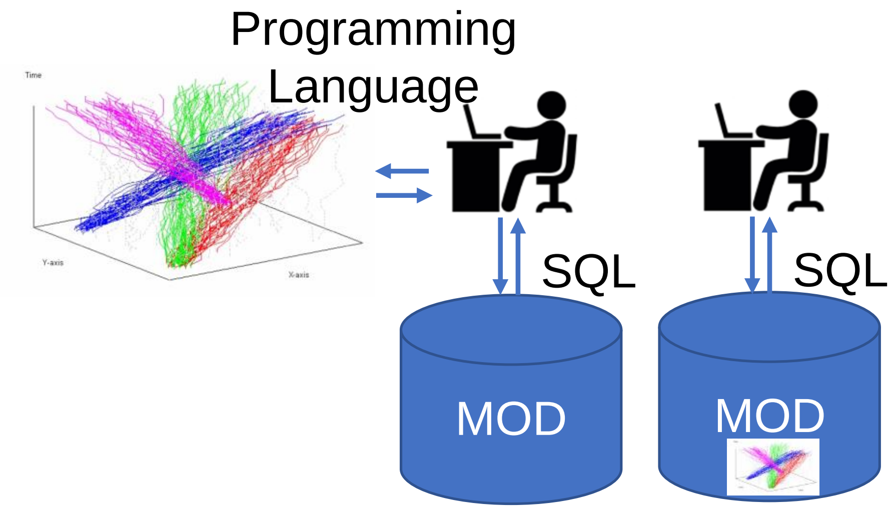
Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Motivation

- The “explosion” of mobility data generation has posed new challenges in the data management community, in terms of **storage**, **querying**, **analytics** and **knowledge extraction**.
- During the past two decades, the field of **Moving Object Databases** (**MODs**) has emerged for the efficient management (**storage**, **querying** and **indexing**) of such data.
- However, knowledge discovery techniques, such as **cluster analysis**, are not treated as an integral part of **MODs**.
- **Bridge the gap between MOD management and mobility data mining → efficiency** and **ease of use**.

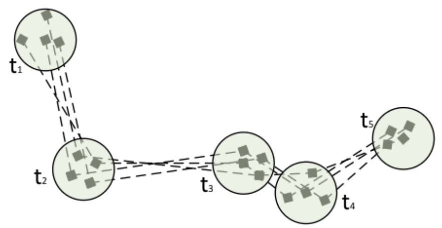
12K distinct ships/day, 200M AIS contacts/month in EU waters



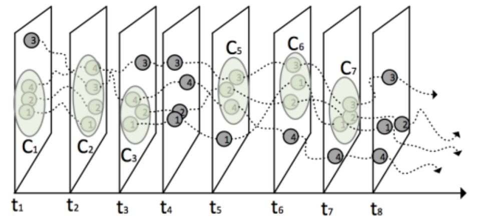
Motivation

- **Trajectory clustering** is an important operation of knowledge discovery from mobility data.
- The research so far has focused mainly in methods that aim to identify specific collective behavior patterns among moving objects.
  - However, this kind of approaches operate at specific predefined temporal "snapshots" of the dataset, thus ignoring the route of each moving object between these sampled points.
- Another line of research tries to identify patterns that are valid for the entire lifespan of the moving objects.
  - However, discovering clusters of complete trajectories can overlook significant patterns that might exist only for some portions of their lifespan.

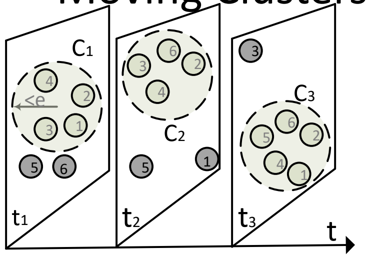
Flocks



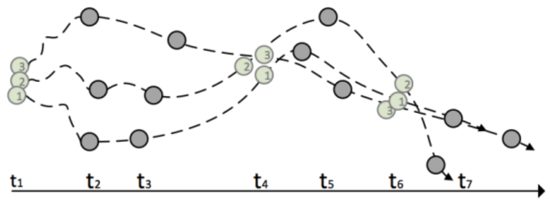
Convoys



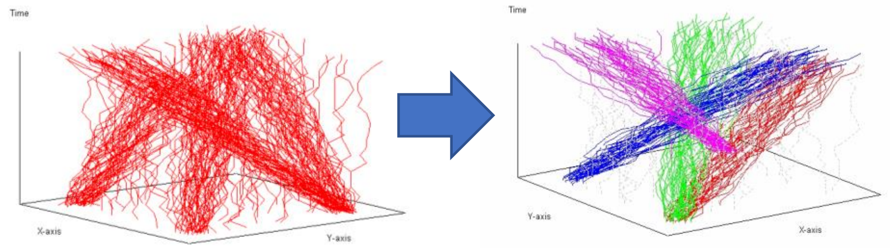
Moving Clusters



Swarms

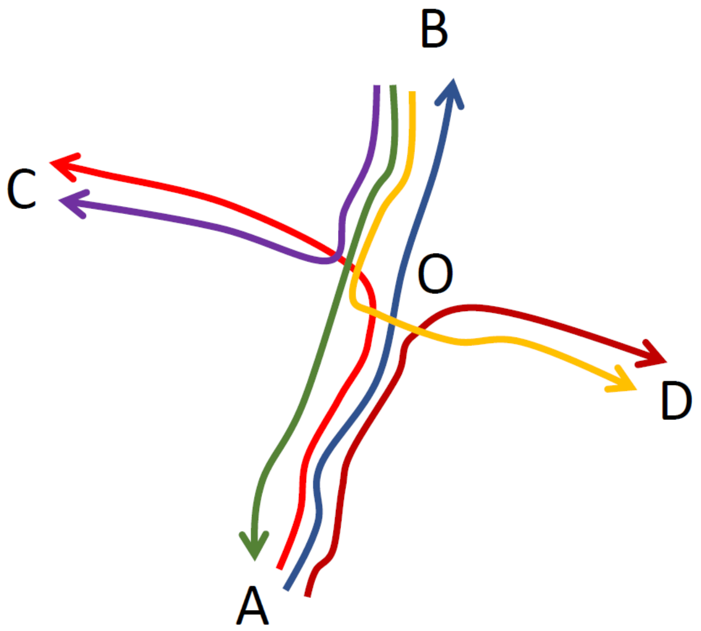


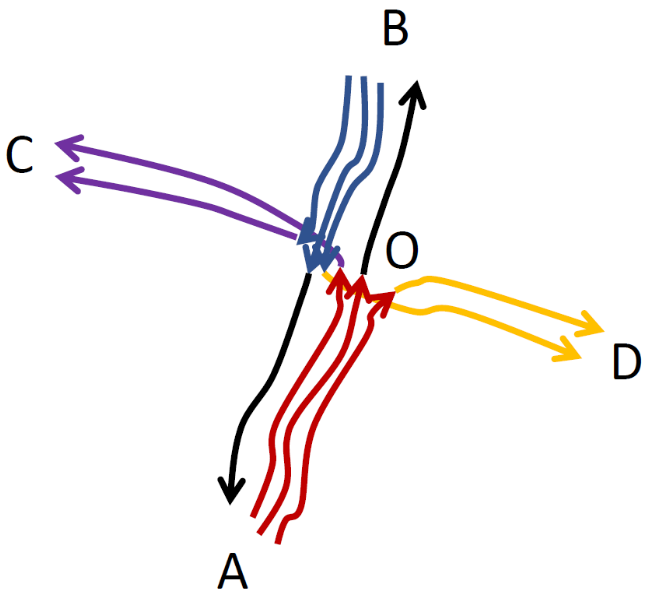
T-OPTICS



Motivation

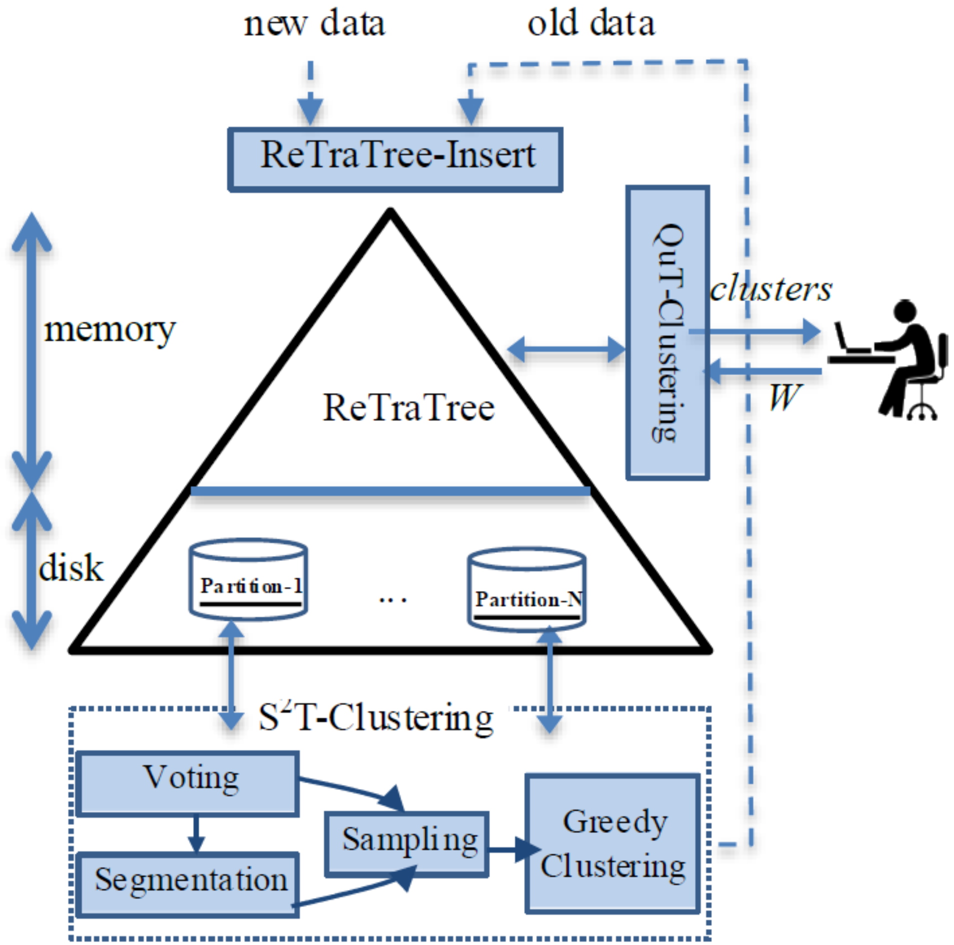
- In this thesis, we focus in **Subtrajectory Clustering analysis**.
- Six Trajectories
  - *A → B*
  - *A → C*
  - *A → D*
  - *B → A*
  - *B → C*
  - *B → D*
- The Goal:
  - 4 Clusters
    - *A → O (red)*
    - *B → O (blue)*
    - *O → C (purple)*
    - *O → D (orange)*
  - *and 2 outliers*
    - *O → A*
    - *O → B (black))*





Motivation

- What is even more challenging, is how one can support incremental and progressive cluster analysis in the context of dynamic applications, where
  - new trajectories arrive at frequent rates, and
  - the analysis is performed over different portions of the dataset, and this might be repeated several times per analysis task.

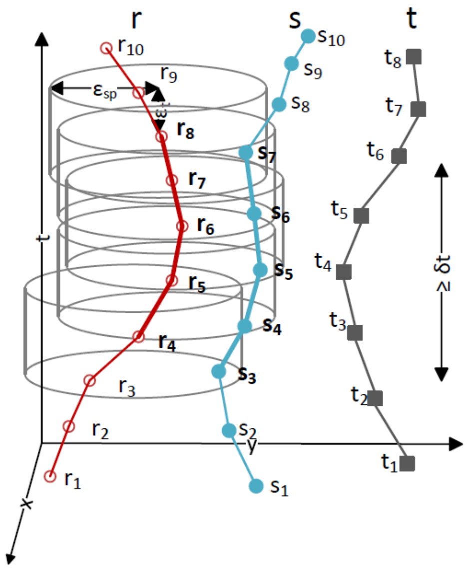


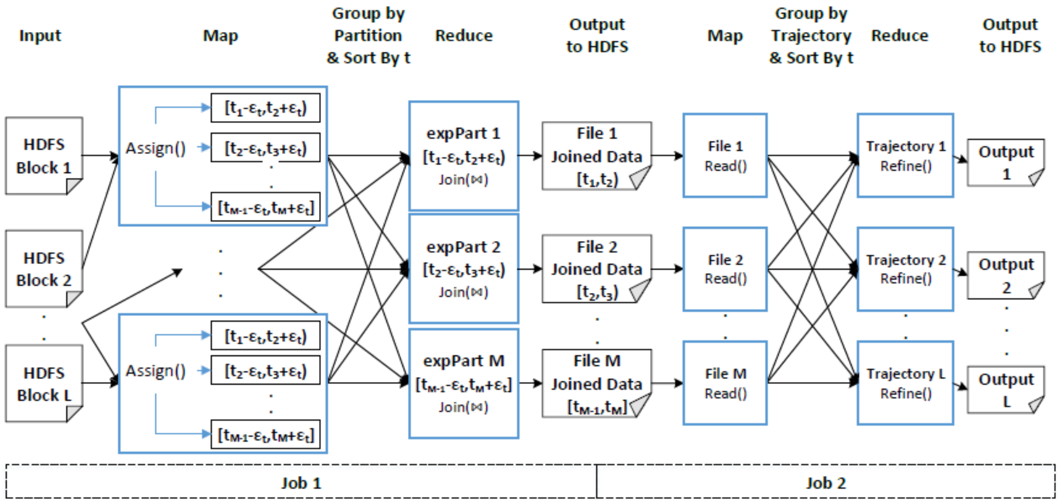
Motivation

- Performing advanced knowledge discovery operations, over immense volumes of data in a centralized way is far from straightforward.
- The bottleneck → spatiotemporal similarity join query

  → Parallel and Distributed algorithms → Scalability + Efficiency.

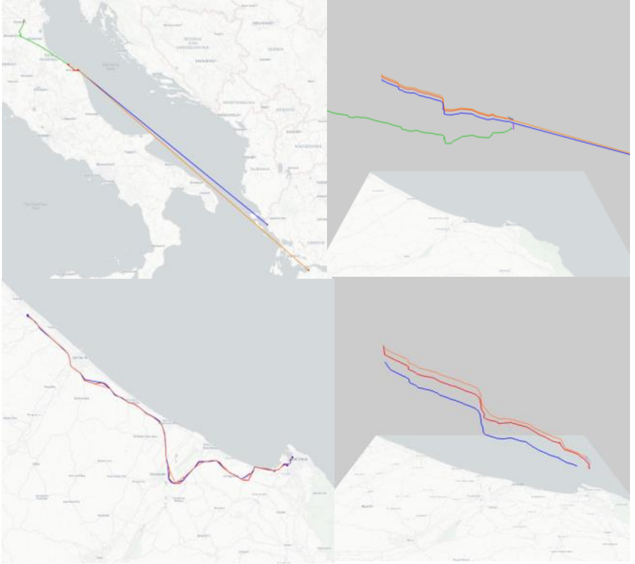
- Joining trajectory datasets is a significant operation with a wide range of applications.



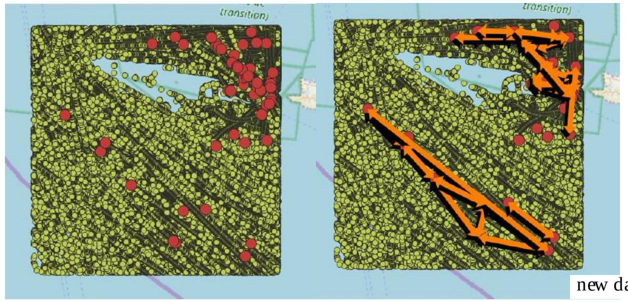


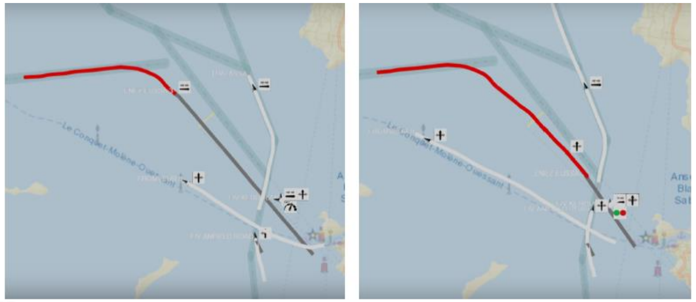
Application Scenarios

- **Trajectory Join**
  - Carpooling; Suspicious Movement Detection; Trajectory segmentation; etc.

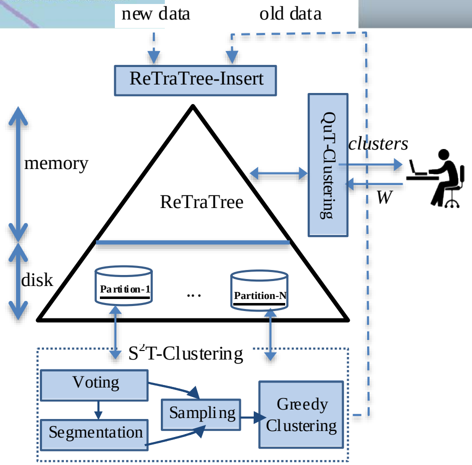


- **Subtrajectory Clustering**
  - Network Discovery; Predictive Analytics; etc.





- **Interactive Mobility Data Exploration and Analysis**
  - Urban Planning; Traffic Analysis; etc.



Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Challenges

- The problem of subtrajectory clustering is NP-Hard.
  - The objects to be clustered are not known beforehand but have to be identified through a trajectory segmentation procedure.

- Implementing efficiently such an algorithm “inside” an extensible DBMS is also a challenging task, since its peculiarities need to be taken into account.

- Parallel and Distributed processing.
  - How to **partition** the data in such a way so that each node can perform its **computation independently**.
  - How to achieve **load balancing**.
  - How to **minimize** the **iterations** of data processing.

Contributions

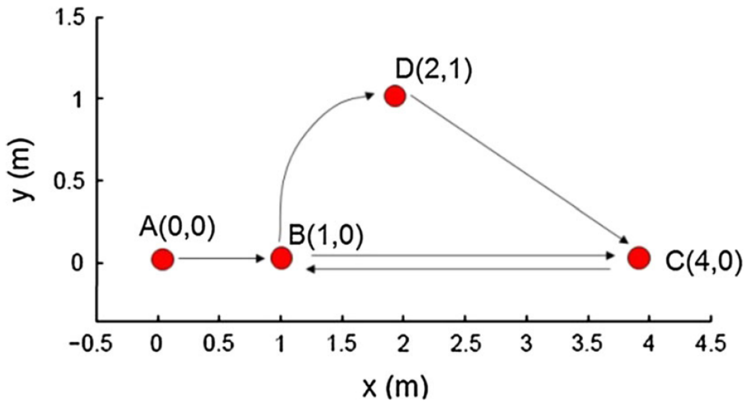
- **In-DBMS Centralized Algorithms and Techniques** (Part II)
  - In-DBMS Sampling-based SubTrajectory Clustering (S^2T-Clustering) (Chapter 3)
  - Temporal-constrained Subtrajectory Cluster Analysis (Chapter 4)
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL (Chapter 5)
- **Distributed Algorithms and Techniques** (Part III)
  - Distributed Subtrajectory Join on Massive Datasets (Chapter 6)
  - Scalable Distributed Subtrajectory Clustering (Chapter 7)

Outline

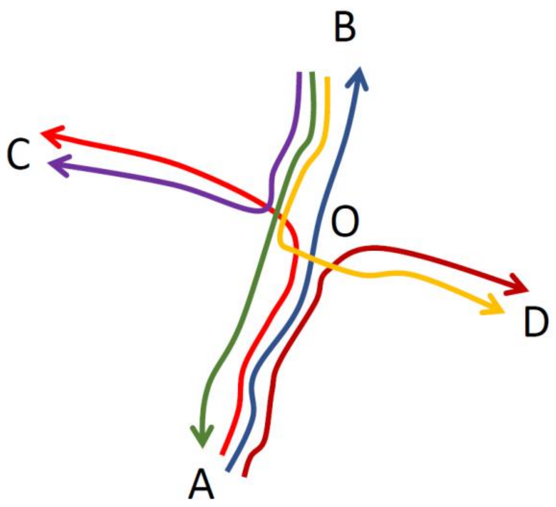
- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Datasets

- Synthetic
  - SMOD - Synthetic MOD (SMOD)



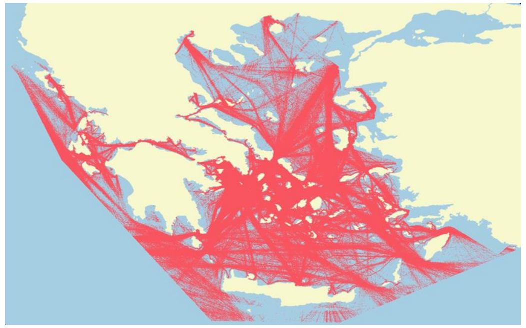
  - Intersection

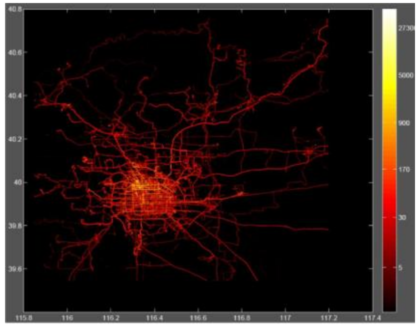


Statistic	SMOD	Intersection
# Trajectories	400	409
# Points	35273	2573
Dataset Duration	120 seconds	23 seconds

Datasets

- Real





Statistic	# Trajectories	# Points	Area	Dataset Duration
IMIS	699031	1.5×10^9	Eastern Mediterranean	3 years
IMIS_1	5110	443657	Greece	1 week
IMIS_2	5110	449680	Eastern Mediterranean	1 week
Brest	365000	17×10^6	Brest	6 months
GeoLife	18668	24×10^6	China and USA	4 years
SIS	2.2×10^7	7.2×10^8	Rome and Tuscany	2.5 years
London Landing	1118	95396	London	1 day

maritime

urban

aviation

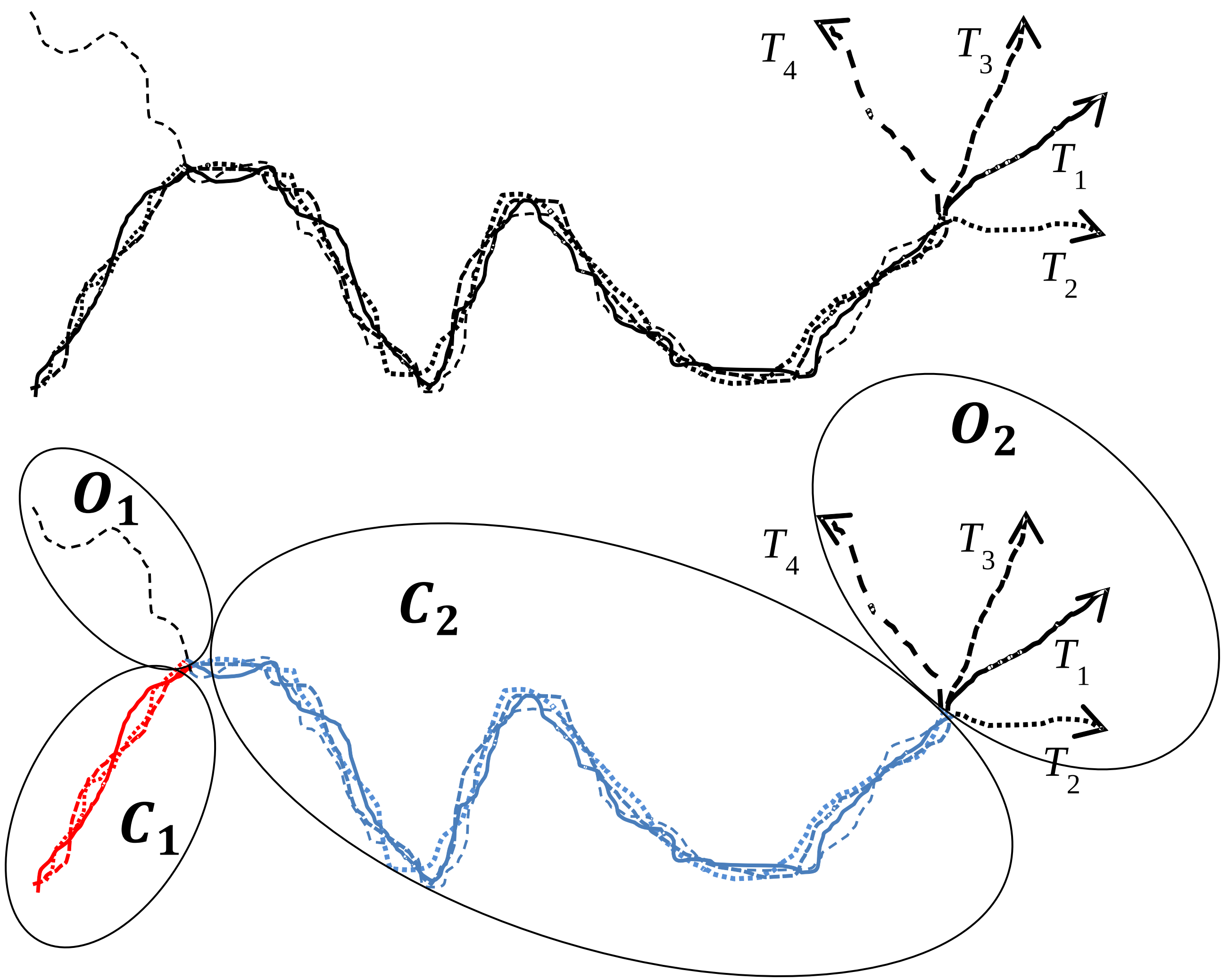
Part II
In-DBMS Centralized Algorithms and Techniques

Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

In-DBMS Sampling-based Subtrajectory Clustering

Introduction



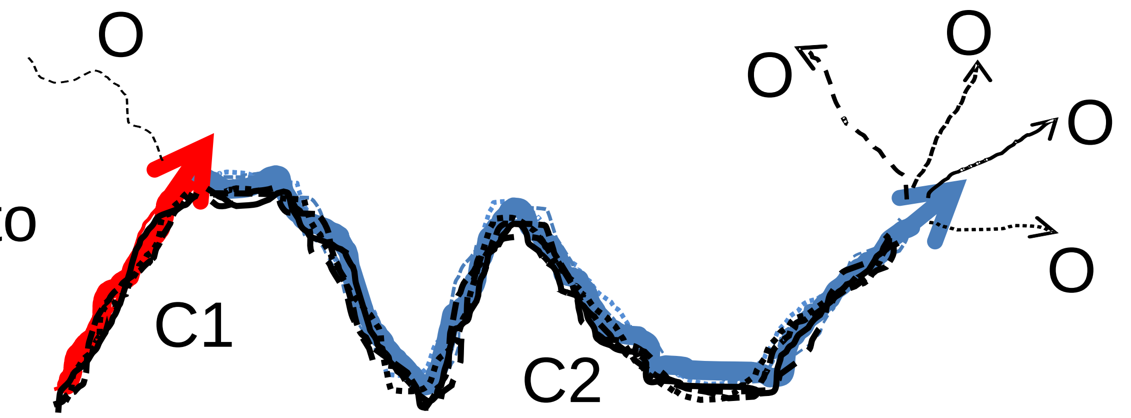
In-DBMS Sampling-based Subtrajectory Clustering

Problem Formulation

- Assuming a cluster is represented by its representative (or medoid) subtrajectory, we define clustering as an optimization:

$$SRD = \sum_{\forall R_j \in S} \sum_{\forall P_{k,i} \in C(R_j)} \overline{V(P_{k,i}, R_j)}$$

- Maximizing *SRD* is not trivial since one has to define, among others,
   i. the criterion according to which a trajectory is segmented into subtrajectories,
   ii. the technique for selecting the set of the most representative subtrajectories,
   iii. whose cardinality M

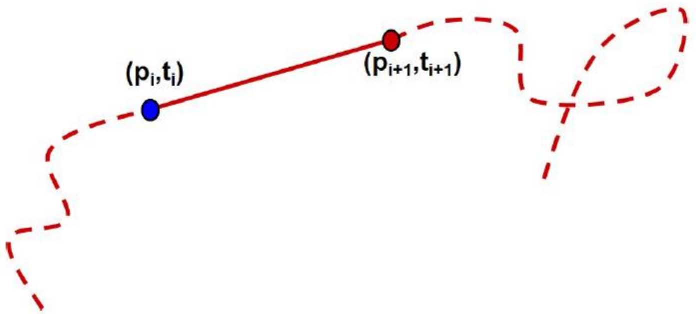


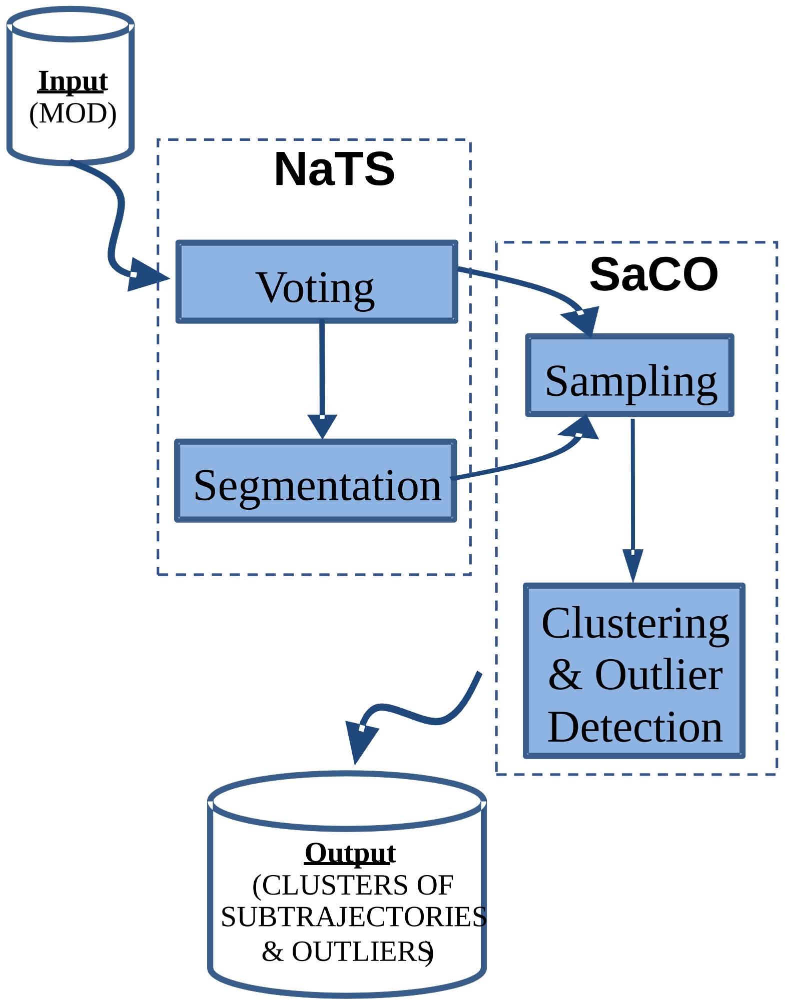
In-DBMS Sampling-based Subtrajectory Clustering

The S^2T-Clustering Algorithm

Algorithm S^2T-Clustering

Input: trajectory dataset $D = \{T_1, T_2, \ldots, T_N\}$, voting influence σ, threshold ϵ
Output: sampling set S, clustering C, set of outliers Out.
// **initialization phase**
Reset set V of voting descriptors in D
// **NaTS phase (Neighborhood-aware Trajectory Segmentation)**
for each trajectory $T_k \in D$ **do**
  Update set V of voting descriptors in D w.r.t. T_k and σ
  Partition T_k in set P_k of subtrajectories w.r.t. V_k
// **SaCO phase (Sampling, Clustering, and Outlier detection)**
Find sampling set S consisting of the M most representative subtrajectories
Using set S and threshold ϵ, partition $P = \cup P_k$ in a set C of M clusters and a set Out of outliers
return (S, C, Out)



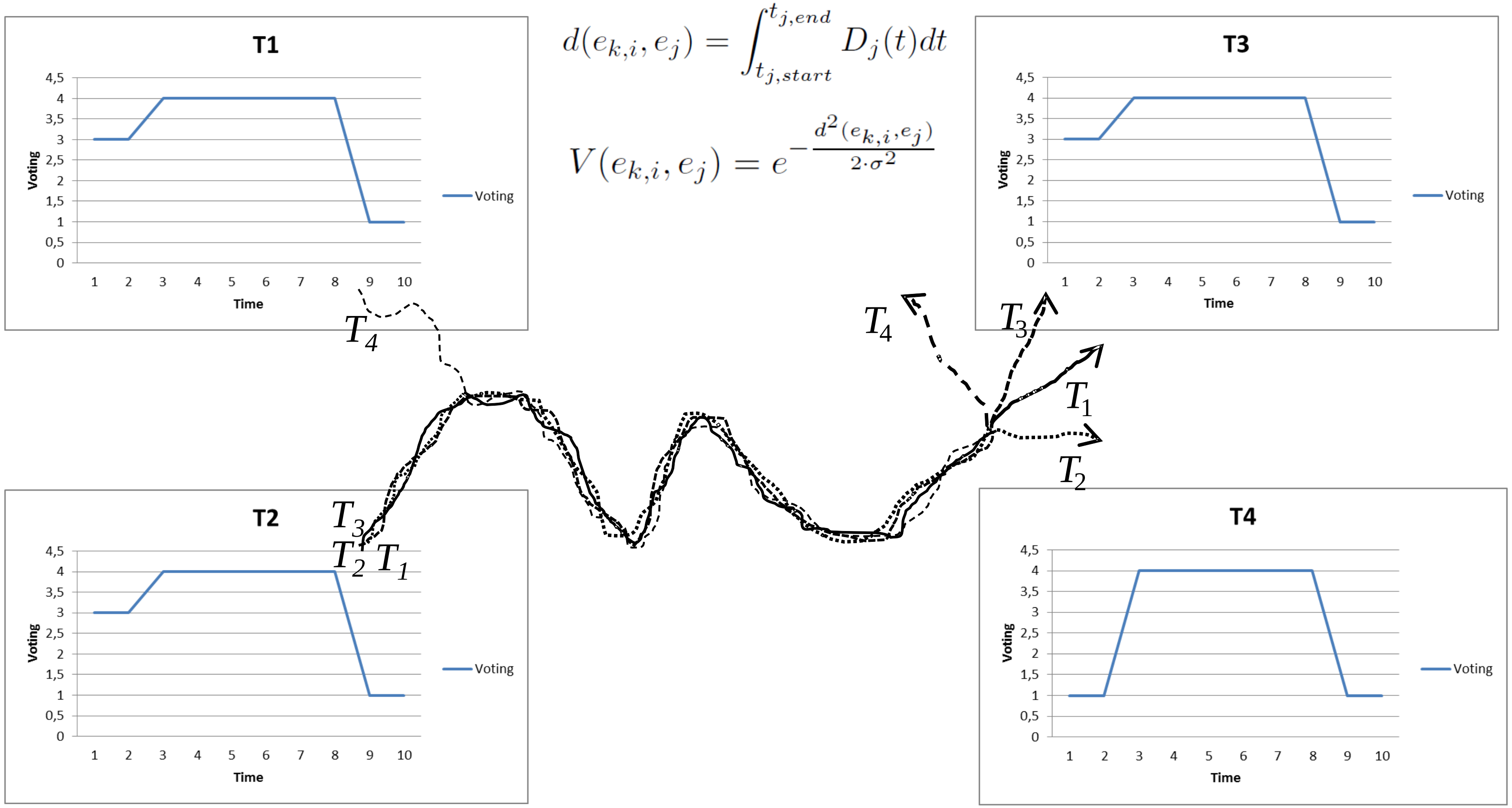


In-DBMS Sampling-based Subtrajectory Clustering

The S²T-Clustering Algorithm - NaTS - Voting

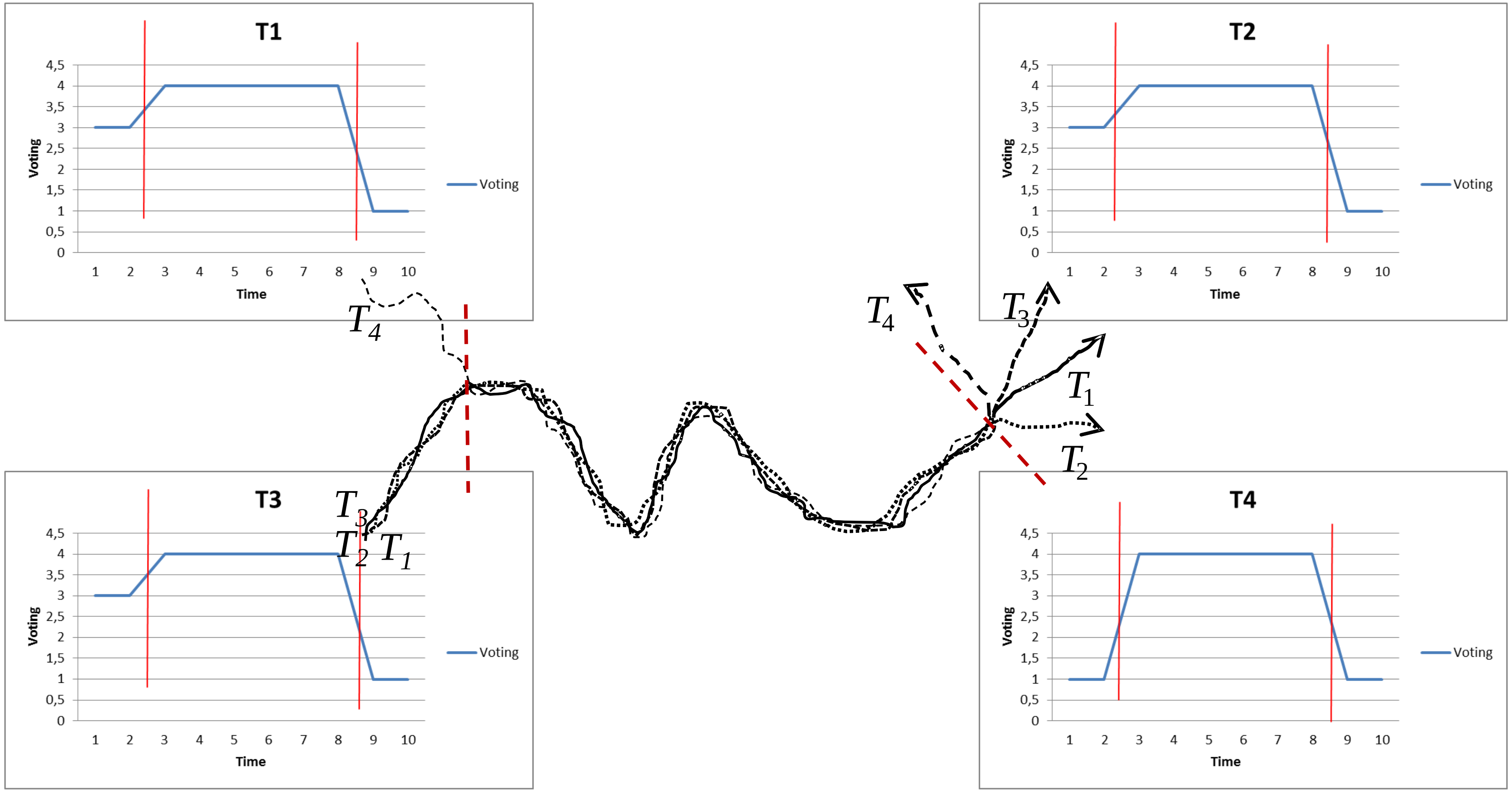
$$d(e_{k,i}, e_j) = \int_{t_{j,start}}^{t_{j,end}} D_j(t)dt$$

$$V(e_{k,i}, e_j) = e^{-\frac{d^2(e_{k,i}, e_j)}{2 \cdot \sigma^2}}$$



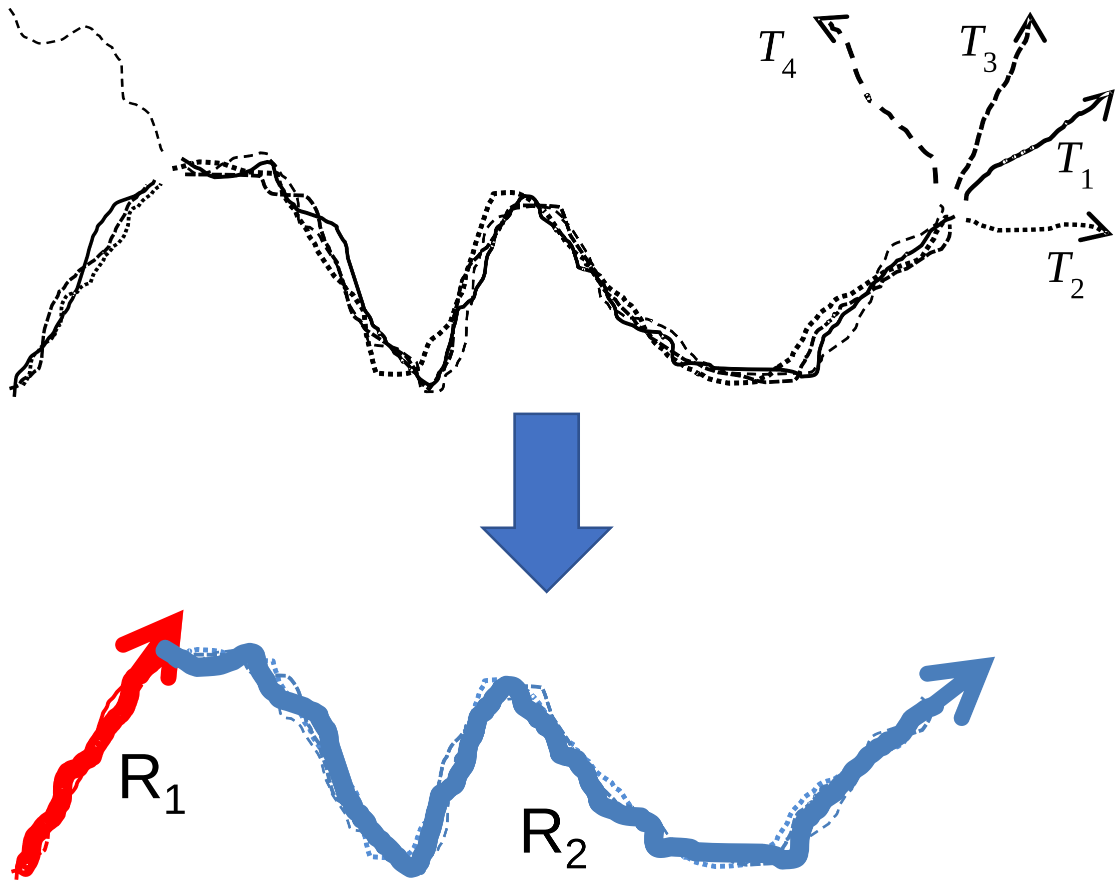
In-DBMS Sampling-based Subtrajectory Clustering

The S^{2}T-Clustering Algorithm - NaTS - Segmentation



In-DBMS Sampling-based Subtrajectory Clustering

The S²T-Clustering Algorithm - SaCO - Sampling



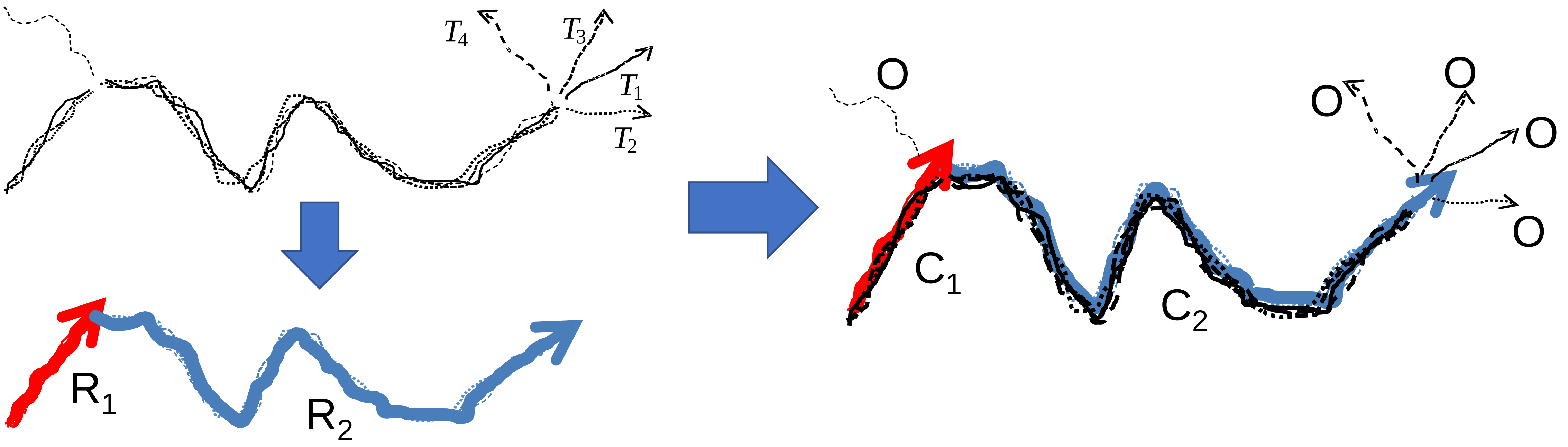
$$SR(S) = \sum_{k=1}^{N} \sum_{i=1}^{LP_k} S_{k,i} \cdot SR_{gain}(k,i)$$

$$SR_{gain}(k,i) = \sum_{j=1}^{|P_{k,i}|} VP^P_{k,i,j} \cdot Nl_{k,i,j} \cdot (1 - VP^S_{k,i,j})$$

$$Nl_{k,i,j} = lifespan(e_{k,i,j}) / lifespanT_k$$

In-DBMS Sampling-based Subtrajectory Clustering

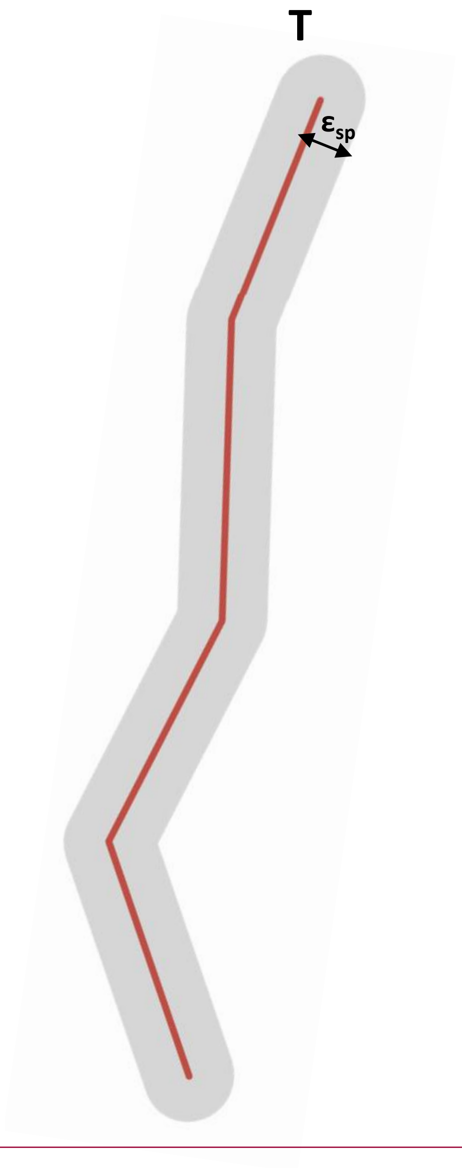
The S^{2}T-Clustering Algorithm - SaCO - Clustering and Outlier detection



In-DBMS Sampling-based Subtrajectory Clustering

S^2T-Clustering in-DBMS

- It is obvious that the most demanding part of the whole procedure is the Voting step.
- Baseline solutions:
  - **Baseline I**: Segment based range query over 3D-R-Tree.
    - One lookup per segment → More disk I/O, better filtering.
  - **Baseline II**: Trajectory based range query over 3D-R-Tree.
    - One lookup per trajectory → Less disk I/O, worse filtering.
- For this reason we implemented the *Trajectory Buffer Query (**TBQ**)* by utilizing the GIST interface.
  - **Trajectory Buffer: TB**(**T**, $\boldsymbol{\varepsilon_{sp}}$, $\boldsymbol{\varepsilon_t}$) → a 3D 'buffer' around **T** such that every point in **TB**(**T**, $\boldsymbol{\varepsilon_{sp}}$, $\boldsymbol{\varepsilon_t}$) is at most $\boldsymbol{\varepsilon_{sp}}$ and $\boldsymbol{\varepsilon_t}$ (in space and time, resp.) far from a point in **T**.



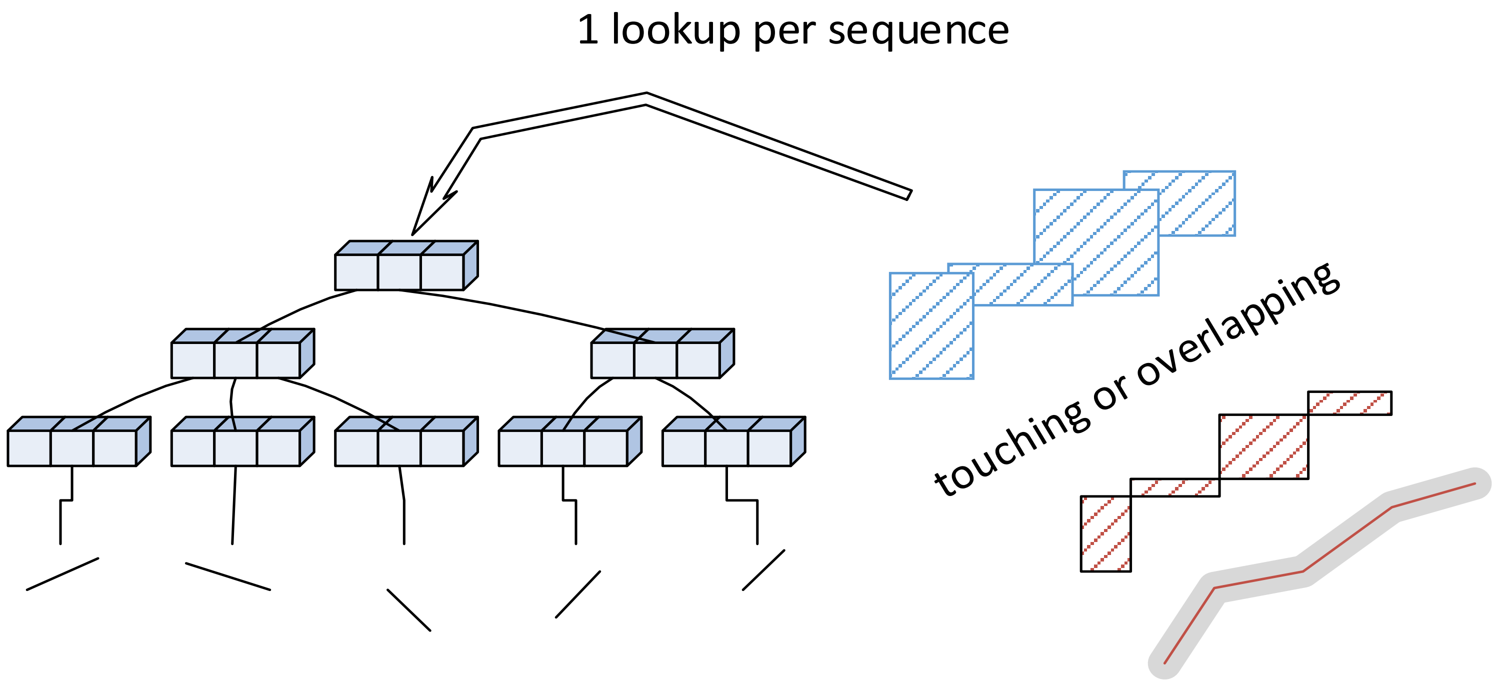
In-DBMS Sampling-based Subtrajectory Clustering

S^2T-Clustering in-DBMS - NaTS in-DBMS

Trajectory Buffer Query - TBQ Given a set **S** of trajectories, a reference trajectory **T**, a spatial threshold $\boldsymbol{\varepsilon_{sp}}$ and a temporal threshold $\boldsymbol{\varepsilon_{t}}$, the trajectory buffer query **TBQ(S, T, $\boldsymbol{\varepsilon_{sp}}$, $\boldsymbol{\varepsilon_{t}}$)** retrieves those segments in S that *overlap* with **TB(T, $\boldsymbol{\varepsilon_{sp}}$, $\boldsymbol{\varepsilon_{t}}$)**. → By implementing GiSTs Consistent() method.

Algorithm 3.3 Consistent

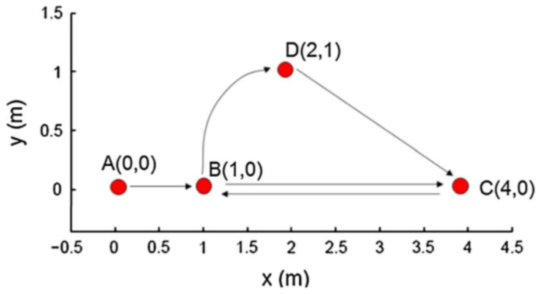
Input: Trajectory Buffer TB_k, current index entry E.
Output: Boolean.
if E is in a leaf node **then**
    if $MBB(E.segment)$ overlaps $MBB(TB_k)$ **then**
        for each $MBB_i \in TB_k$ **do**
            if $E.segment$ overlaps MBB_i **then**
                return true
else
    if $E.box$ overlaps $MBB(TB_k)$ **then**
        for each $MBB_i \in TB_k$ **do**
            if $E.box$ overlaps MBB_i **then**
                return true
    return false

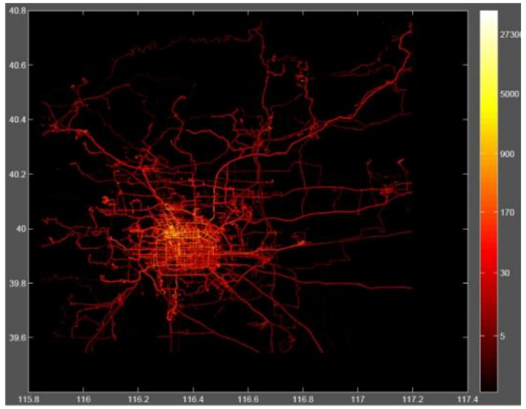


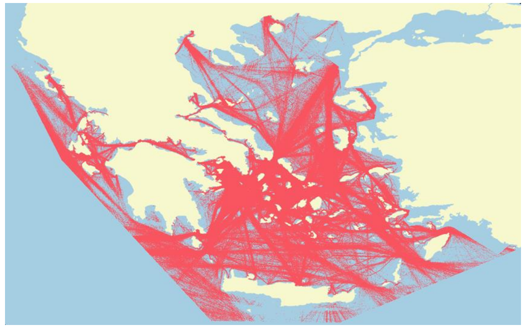
In-DBMS Sampling-based Subtrajectory Clustering

Experimental Study – Datasets

Statistic	SMOD	GeoLife	IMIS$_1$
# Trajectories	400	18668	5110
# Segments	35273	24159325	444570
Dataset Duration (hh:mm:ss)	0:02:00	1932 days 22:59:48	6 days 19:59:53
Avg. Sampling Rate (hh:mm:ss)	0:00:01	0:00:08	0:18:02
Avg. Segment Length (m)	8	72	1545
Avg. Segment Speed (m/s)	7.83	5.01	7.03
Avg. Trajectory Speed (m/s)	2.86	3.91	4.52
Avg. # Points per Trajectory	89	1295	88
Avg. Trajectory Duration (hh:mm:ss)	0:01:28	2:43:15	11:33:45
Avg. Trajectory Length (m)	691	93046	134,148



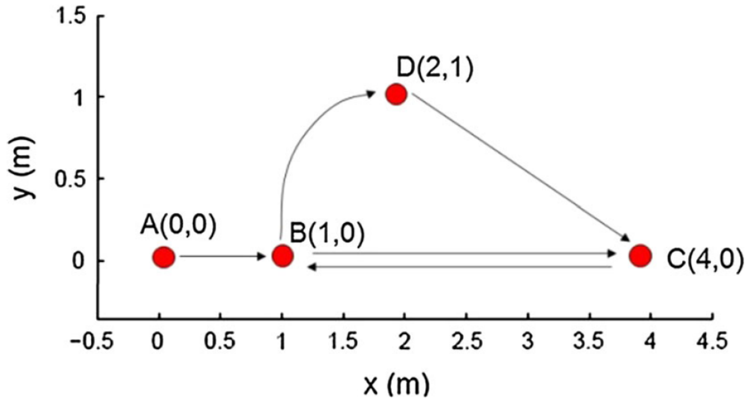




In-DBMS Sampling-based Subtrajectory Clustering

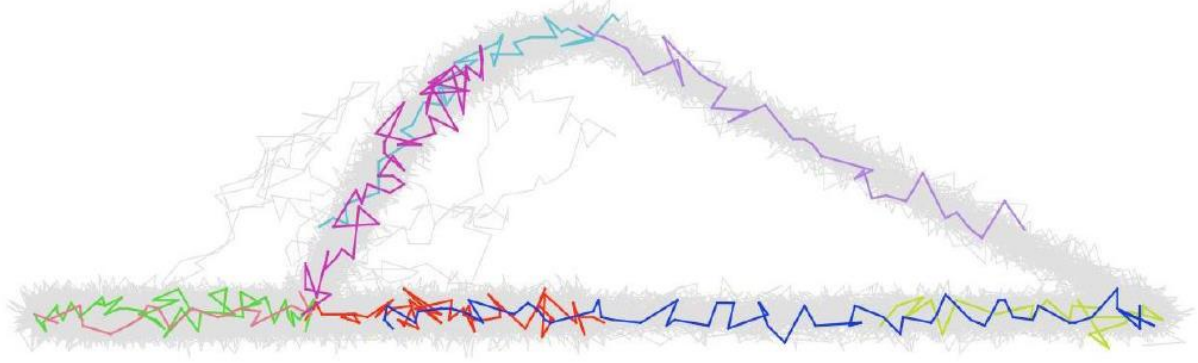
Experimental Study

- S²T-Clustering vs TraClus

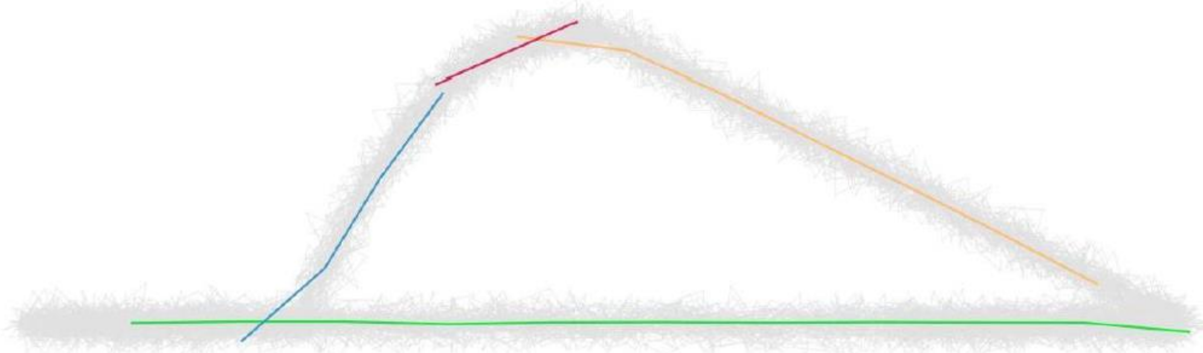


Cluster	Path	Time periods (clusters)
#1, #2	$A \to B$	[0, 0.2], [0.2, 0.7]
#3, #4	$B \to C$	[0.2, 0.8], [0.7, 1.2]
#5, #6	$B \to D$	[0.2, 0.52], [0.7, 1.2]
#7	$C \to B$	[0.8, 1]
#8	$D \to C$	[0.52, 1]

S²T-Clustering

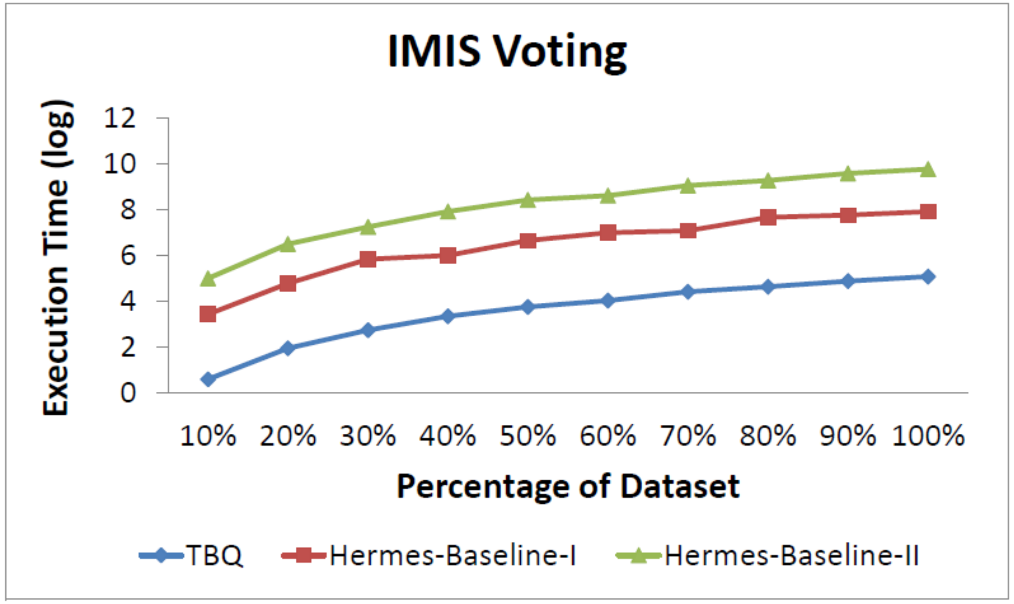


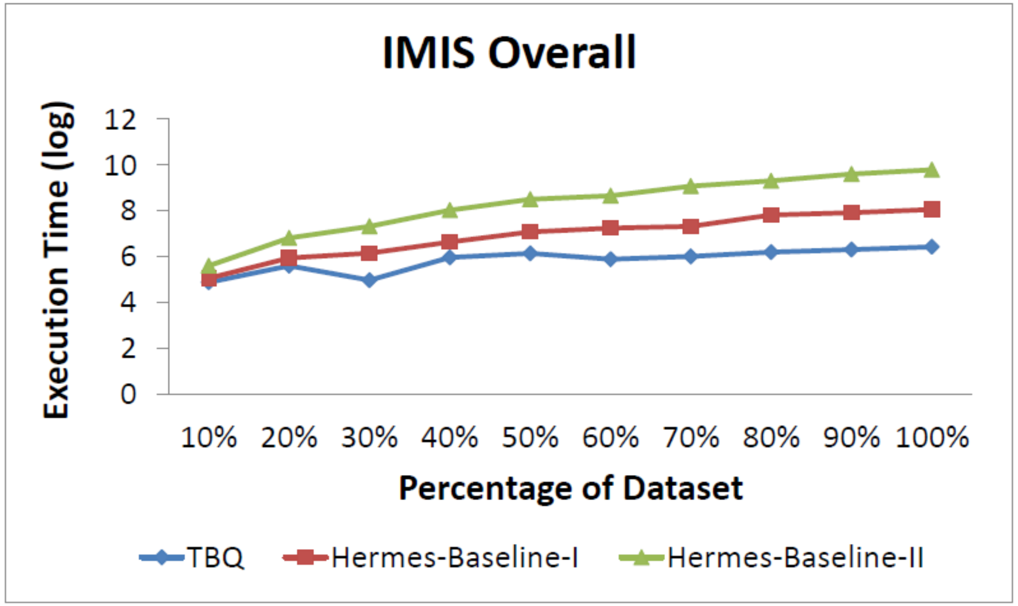
TraClus



In-DBMS Sampling-based Subtrajectory Clustering

Experimental Study - Efficiency and Scalability





In-DBMS Sampling-based Subtrajectory Clustering

Summary

- We address the problem of subtrajectory clustering more effectively than the state-of-the-art (namely, TraClus).
- Our proposal is designed in-DBMS,
  - i.e., it performs as a query operator in a real MOD engine over an extensible DBMS.
- Our algorithm is boosted by an efficient index-based Trajectory Buffer Query (TBQ) that speeds up the overall process,
  - thus resulting in a scalable solution, outperforming the state-of-the-art in-DBMS solutions supported by PostGIS by several orders of magnitude.

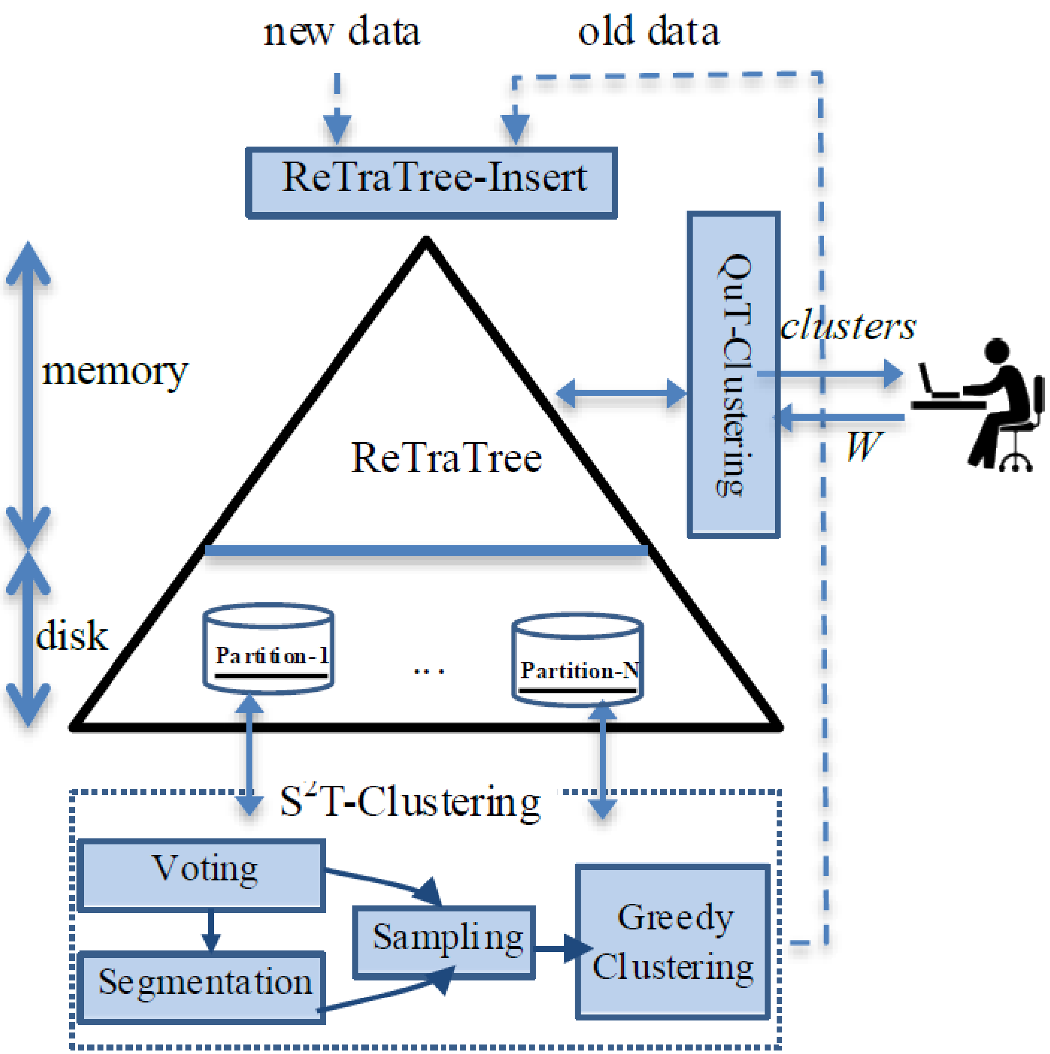
Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Temporal-constrained Subtrajectory Cluster Analysis

Introduction

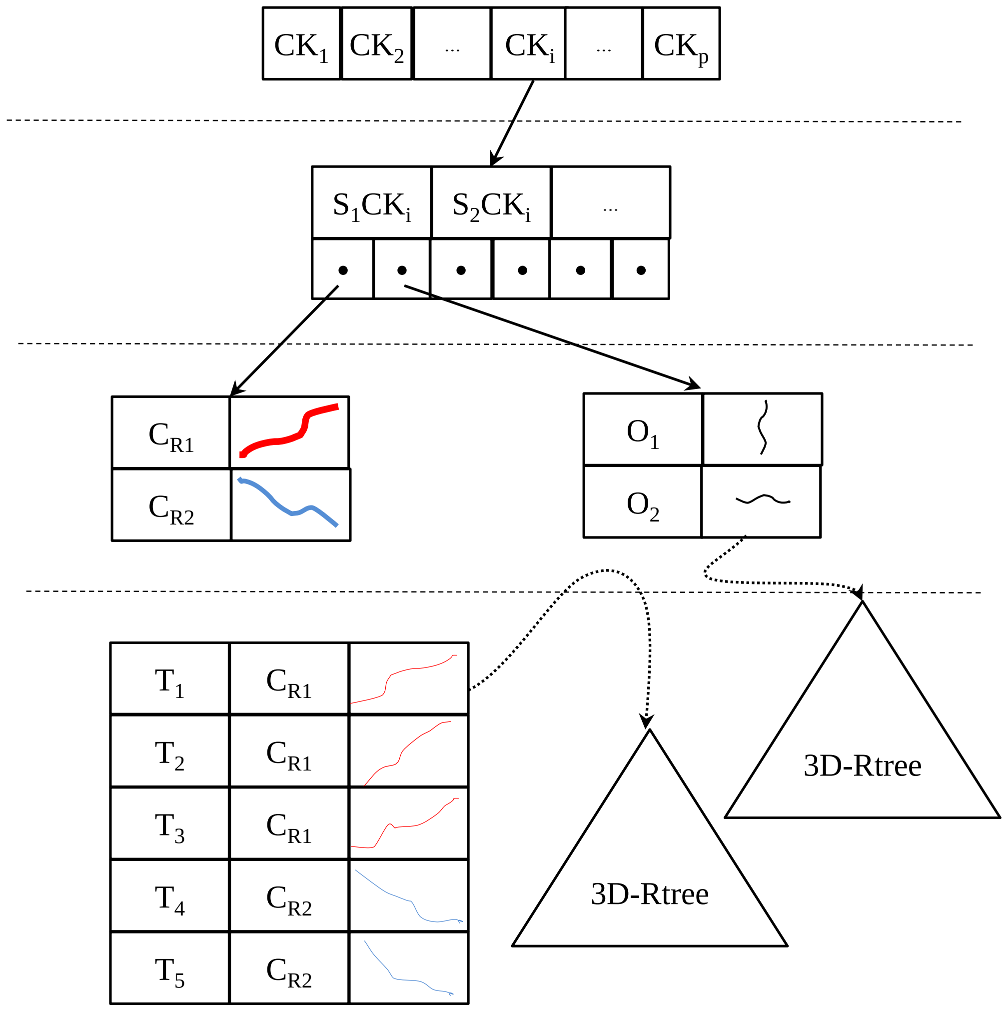
- In several operational applications, new data arrive at frequent rates.
- In real life scenarios, an analyst needs to run the clustering procedure several times and at different portions of a dataset.
- In this setting, the approach followed so far is not that efficient.
- We need a solution that will be able to support efficiently incremental and progressive cluster analysis.



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - Overview

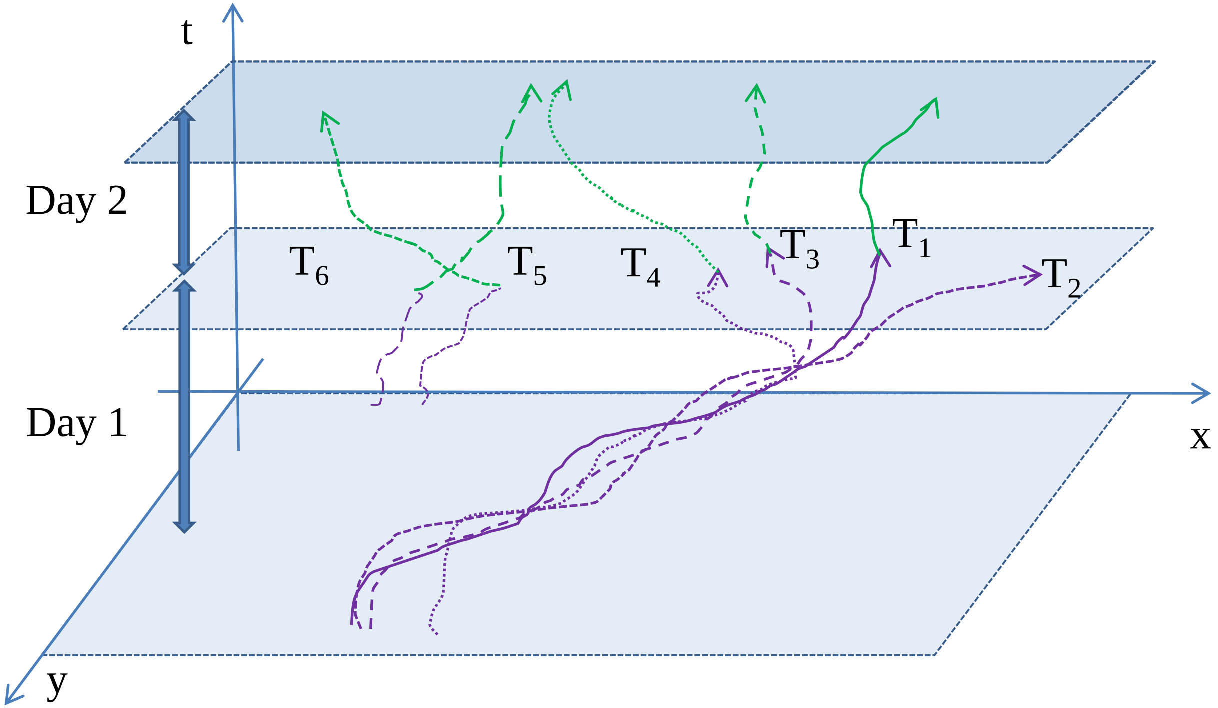
- 1st level - Chunking
- 2nd level - Subchunking
- 3rd level - S^2T Clustering
- 4th level - Raw Data

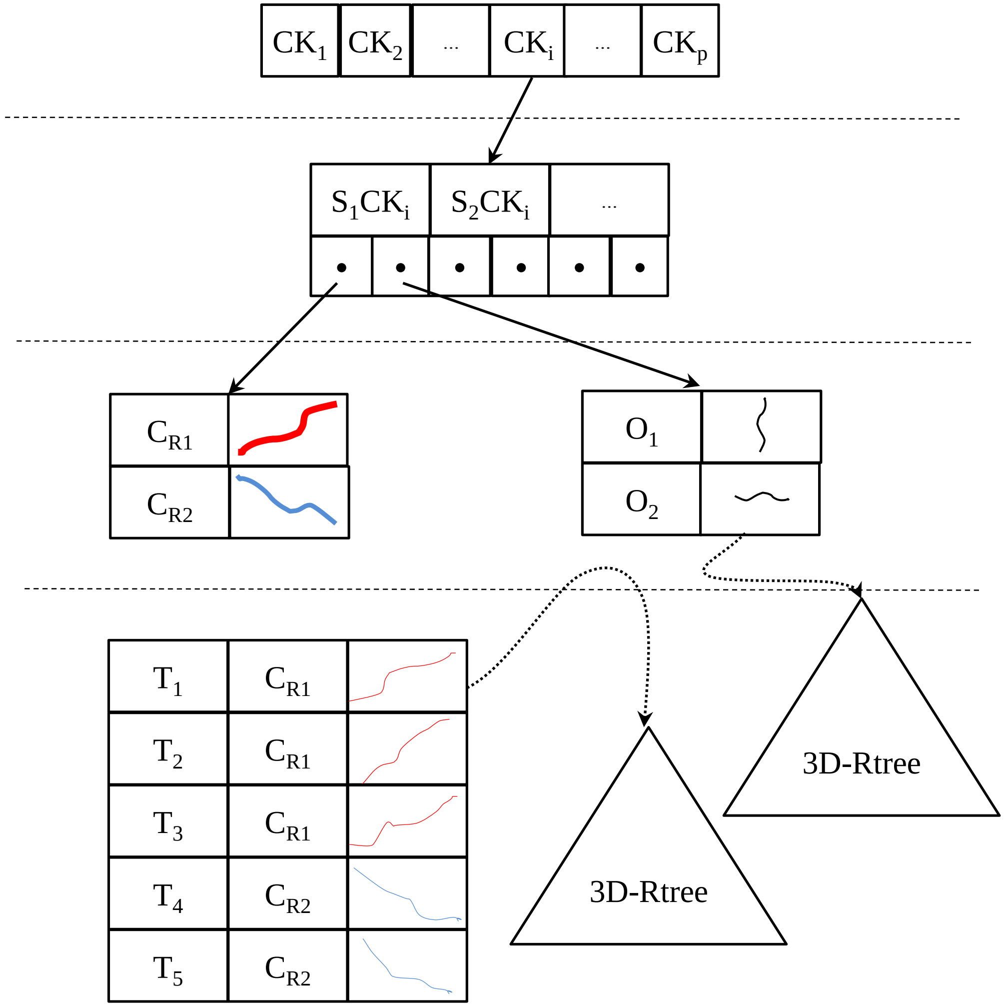


Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - Hierarchical Temporal Partitioning

- 1st level - Chunking

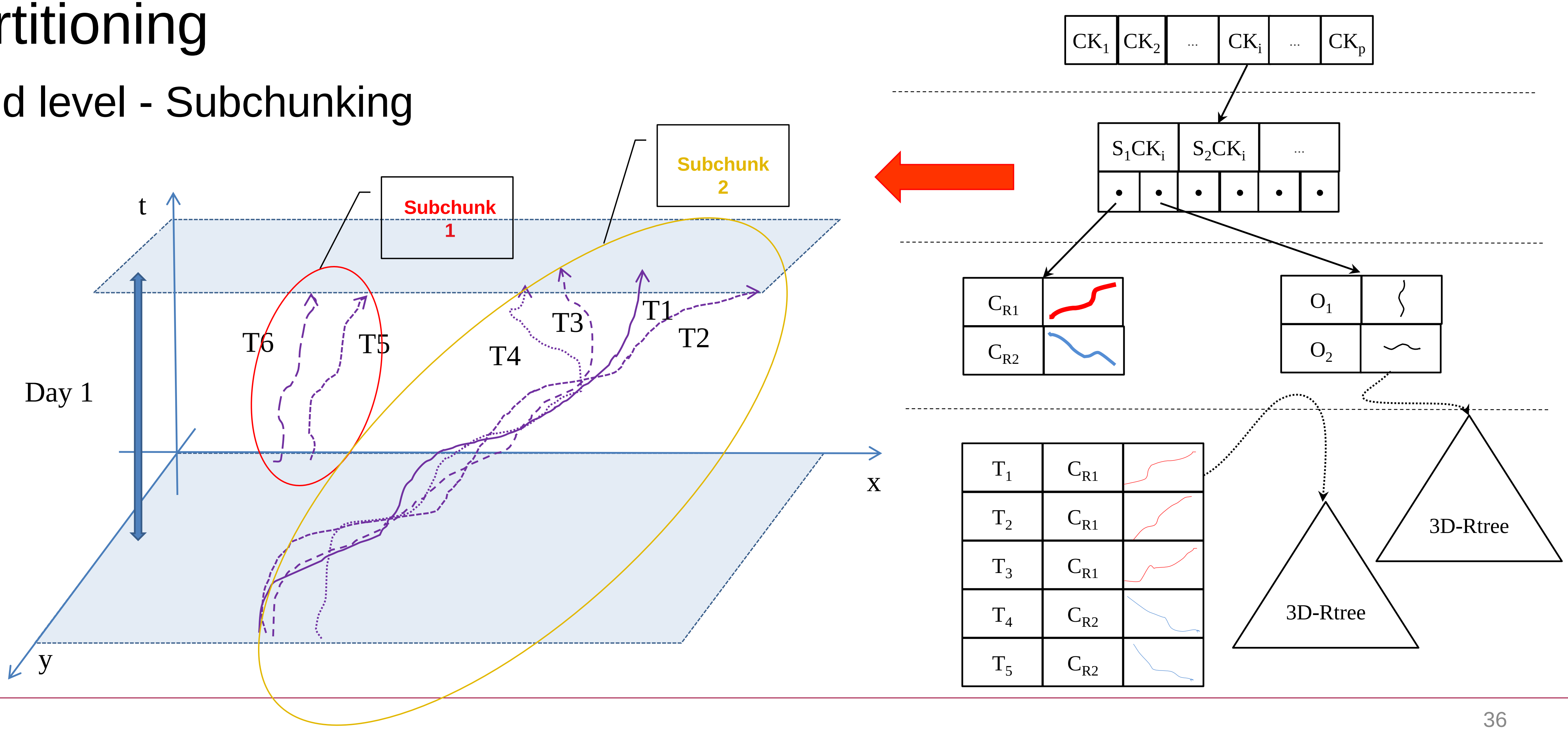




Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - Hierarchical Temporal Partitioning

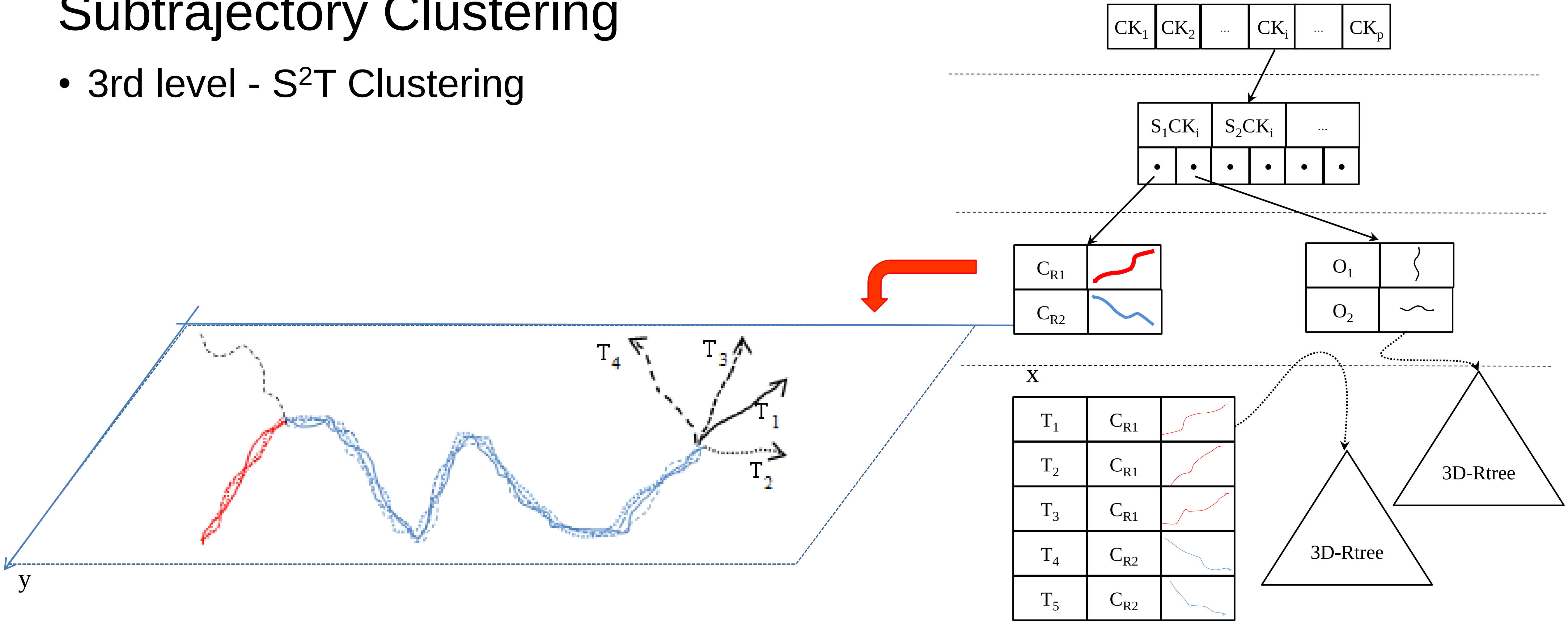
- 2nd level - Subchunking



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - Sampling-based Subtrajectory Clustering

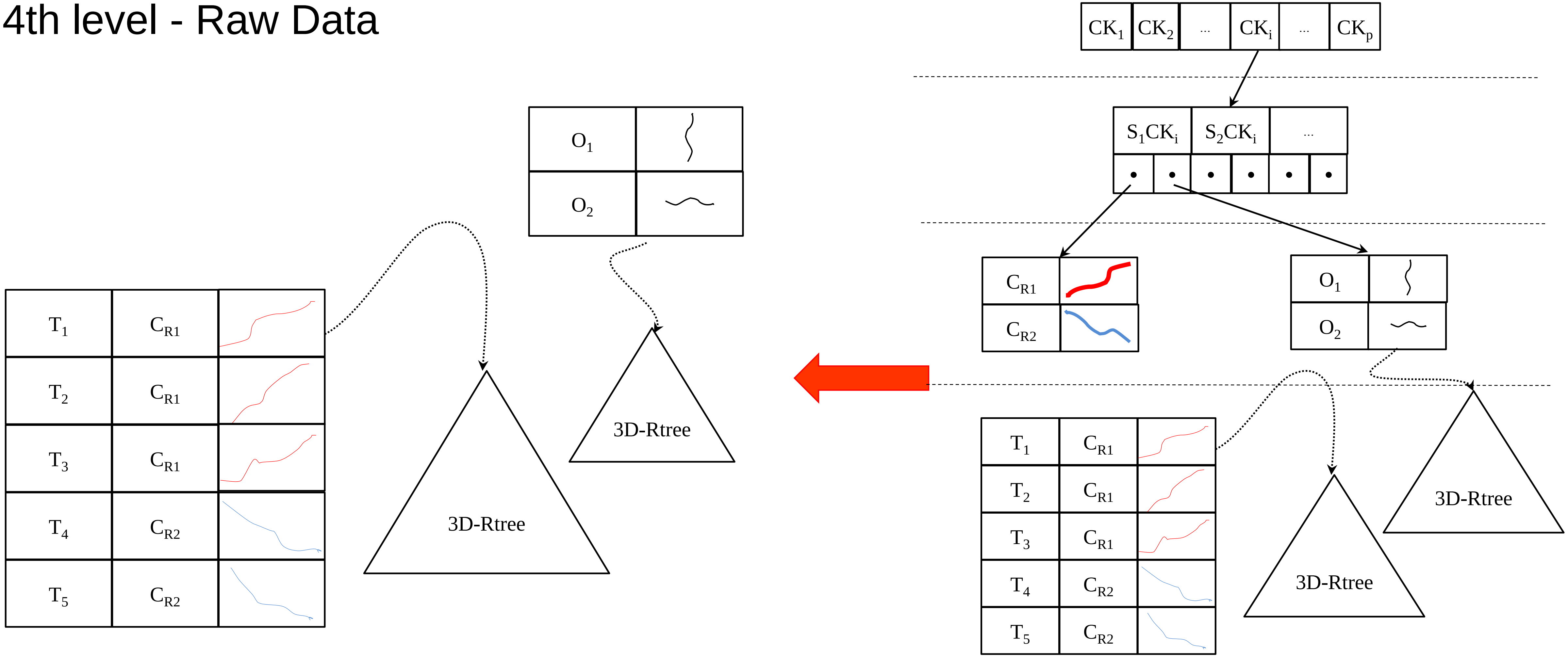
- 3rd level - S^2T Clustering



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme – Raw Data

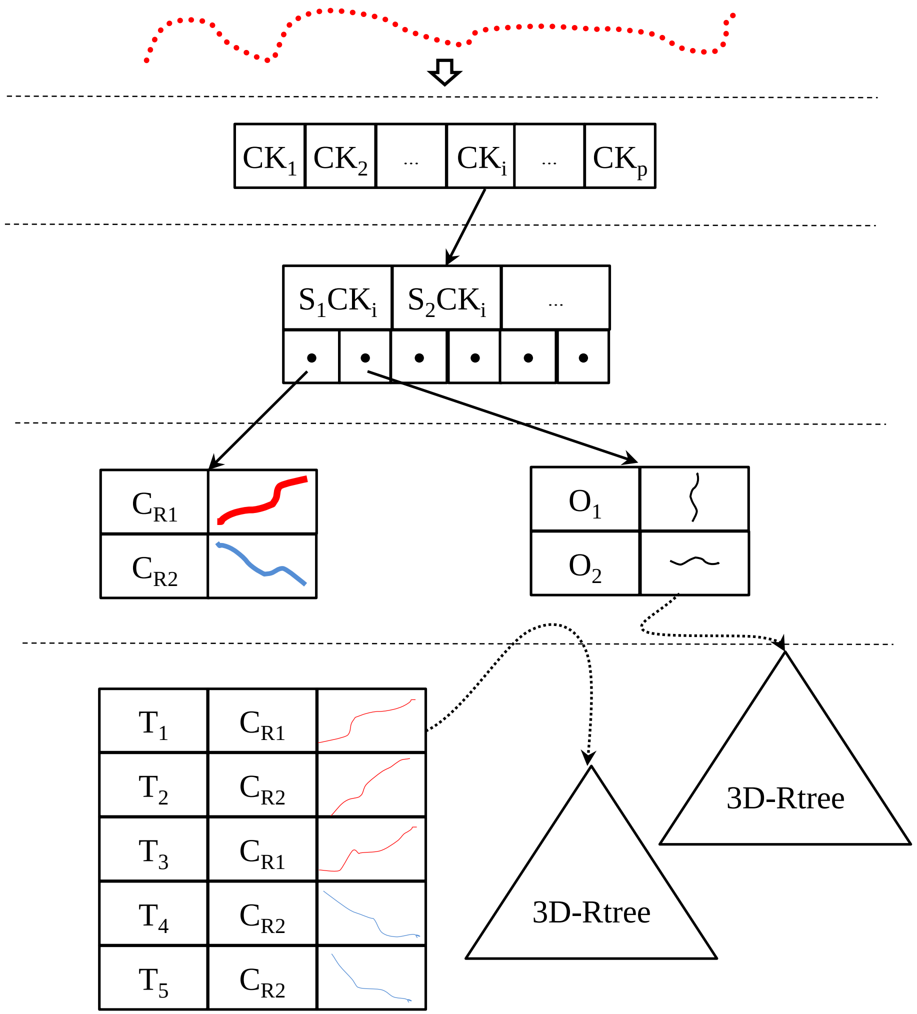
- 4th level - Raw Data



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

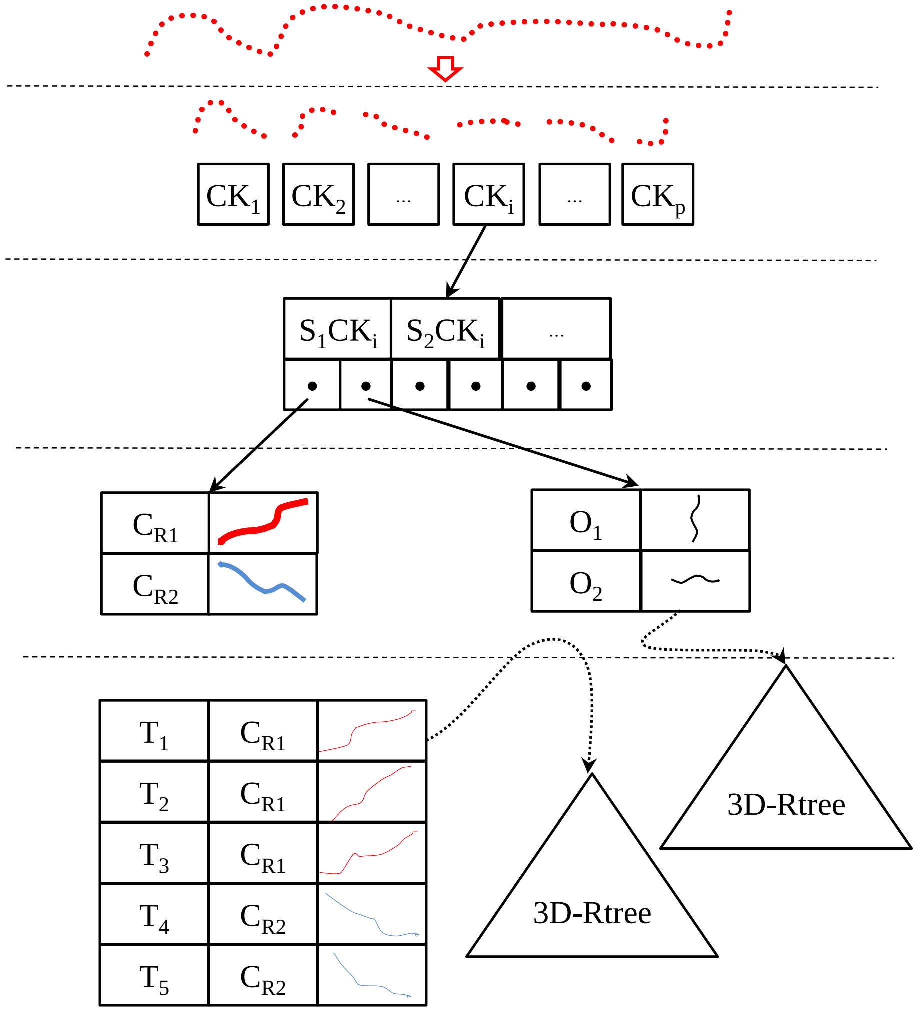
- Adding Trajectory T_k into the ReTraTree Structure.



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

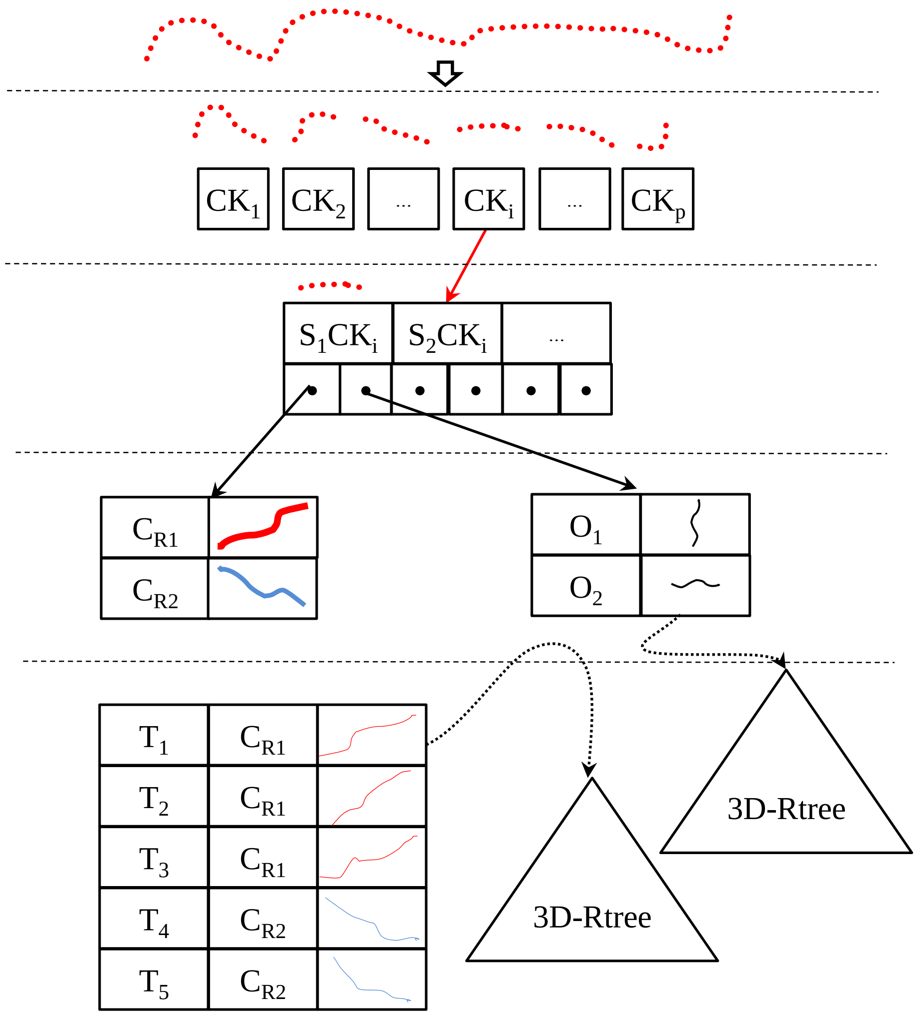
- 1st level Chunking
  - Hard partitioning in the time dimension.



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

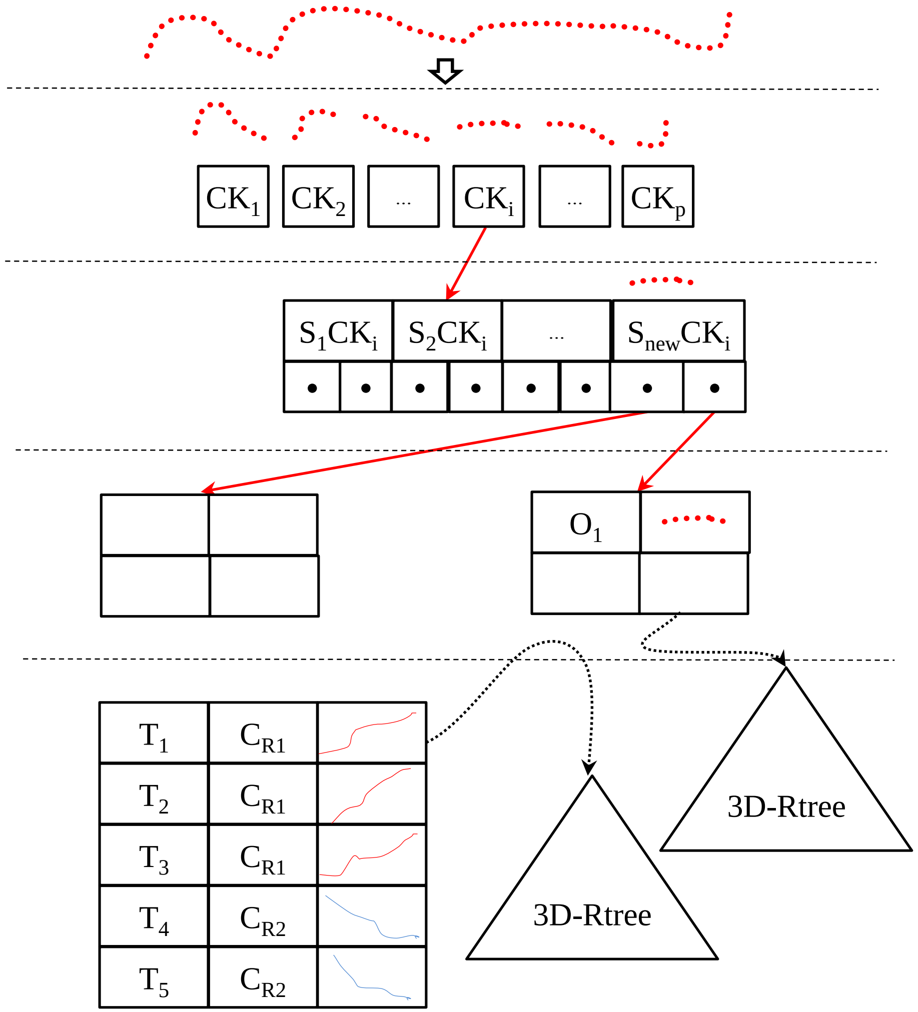
- 2^{nd} level Subchunking
  - Data-driven partitioning in time dimension
  - Assigns each sub-trajectory to an appropriate sub-chunk



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

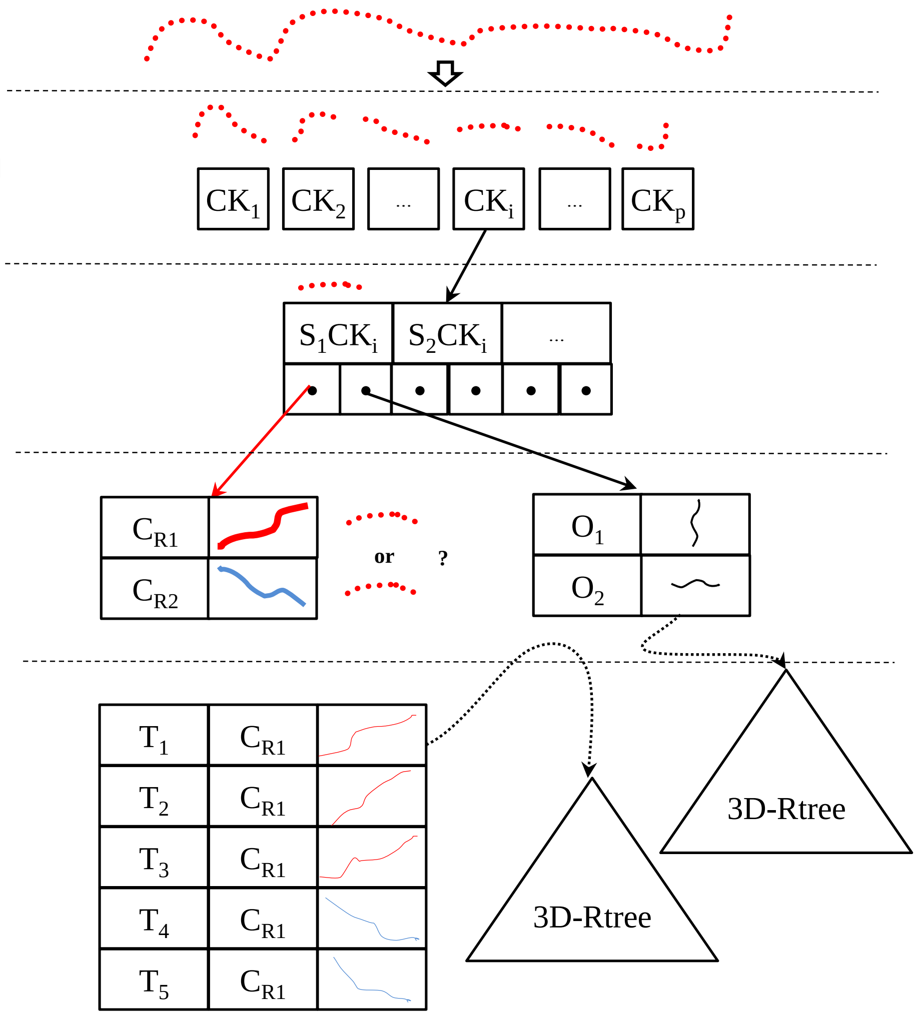
- 2nd level Subchunking
  - If there is not a matching sub-chunk w.r.t. time,
    - a new subchunk is created,
    - which is initialized with an empty representative set *S*,
    - and an outliers set *O* including the unmatched sub-trajectory



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

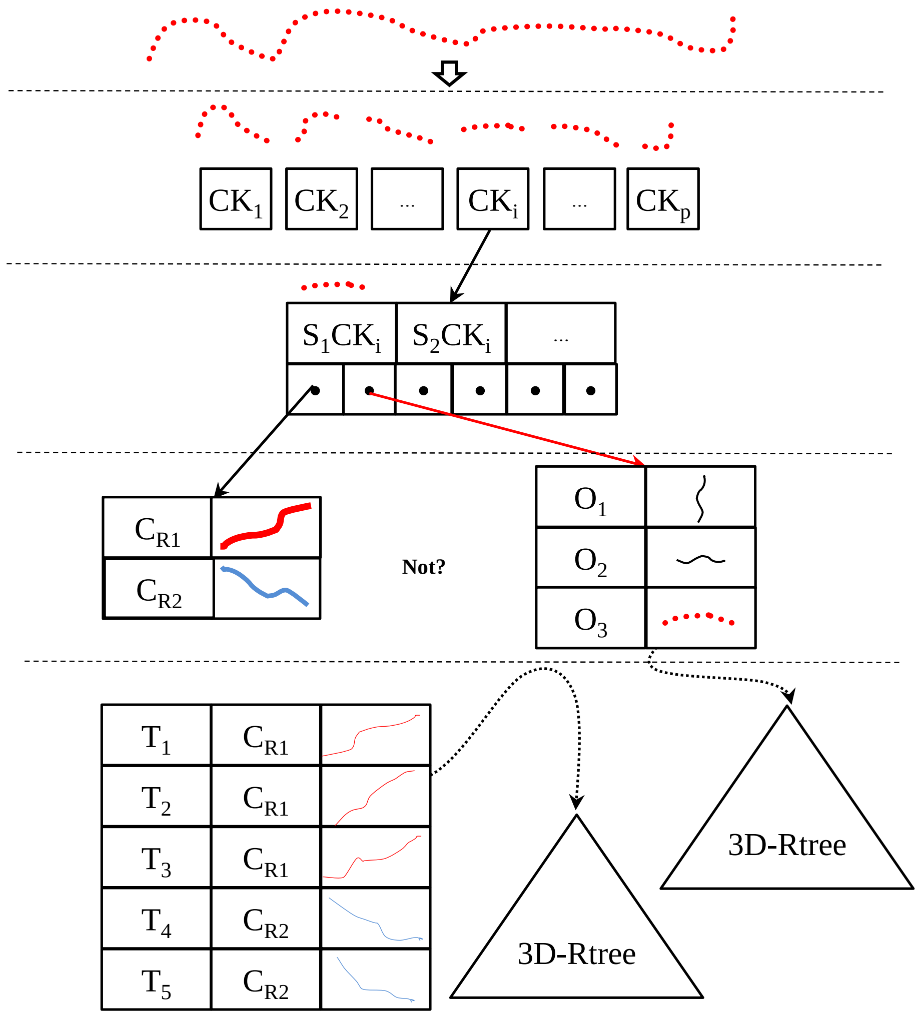
- 3rd level S2T-Clustering
  - If there is an appropriate sub-chunk for the sub-trajectory under processing, the algorithm tries to assign it to an existing cluster.



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

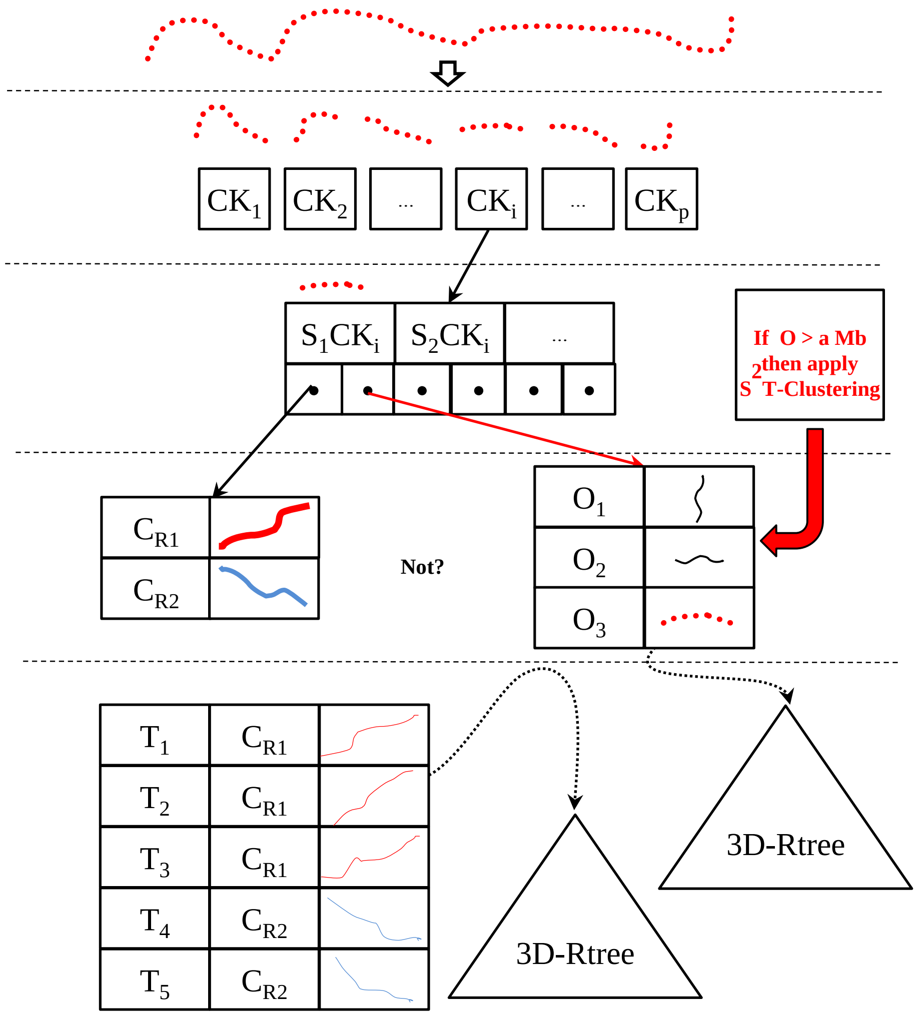
- 3rd level S^2T-Clustering
  - If this attempt fails then the algorithm adds the sub-trajectory into the outliers' set, which act as a temporary relation.



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

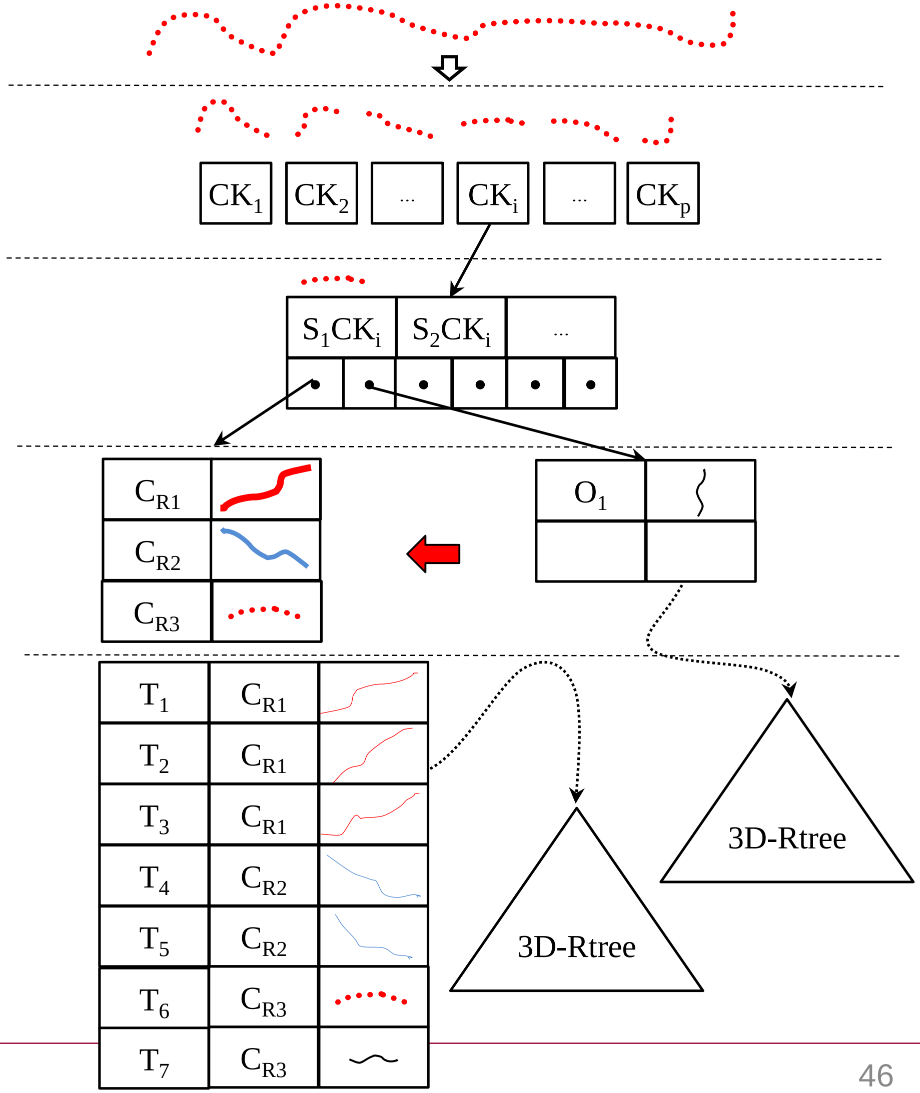
- 3rd level S^2T-Clustering
  - If the size of the relation outliers exists a user-defined threshold set, then sampling-based sub-trajectory clustering is applied.



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

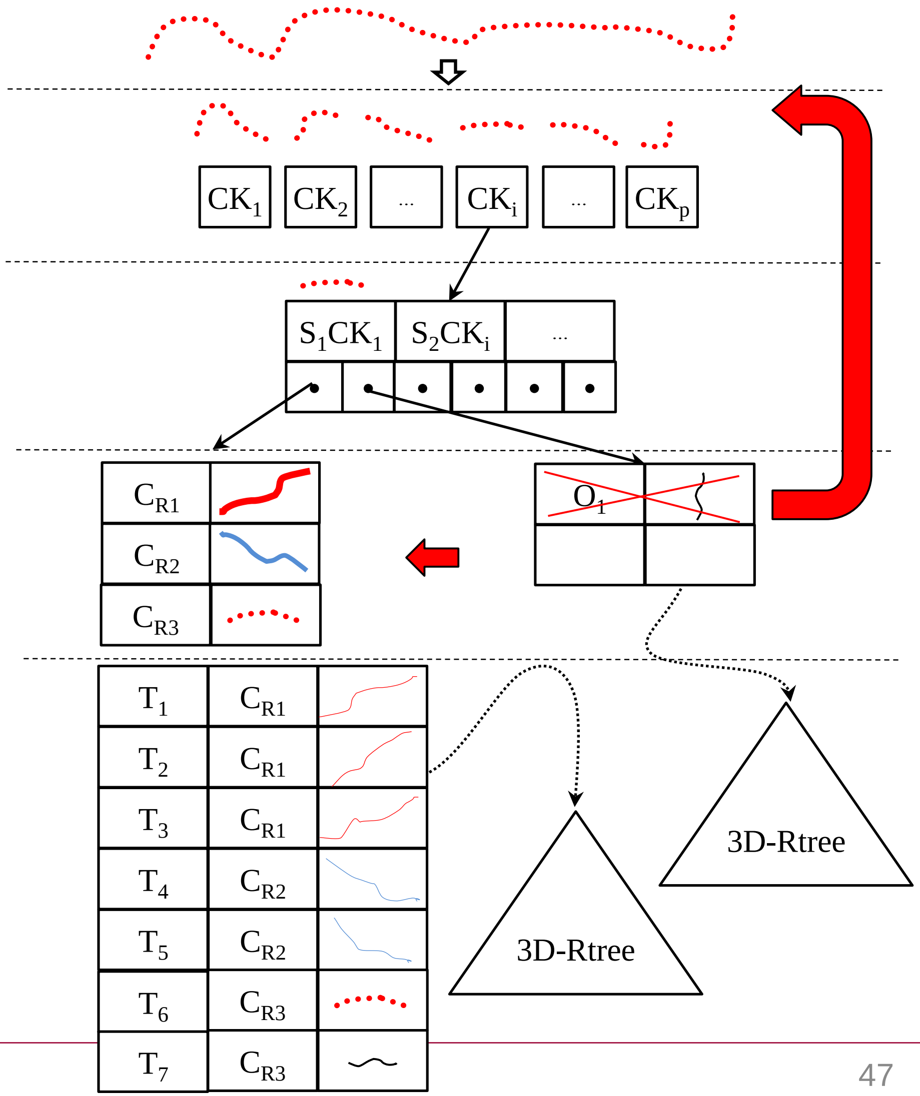
- 3rd level S^2T-Clustering
  - Let us assume that outliers O_2 and O_3 form a cluster and O_1 continues to be an outlier.



Temporal-constrained Subtrajectory Cluster Analysis

The ReTraTree Indexing Scheme - ReTraTree Maintenance

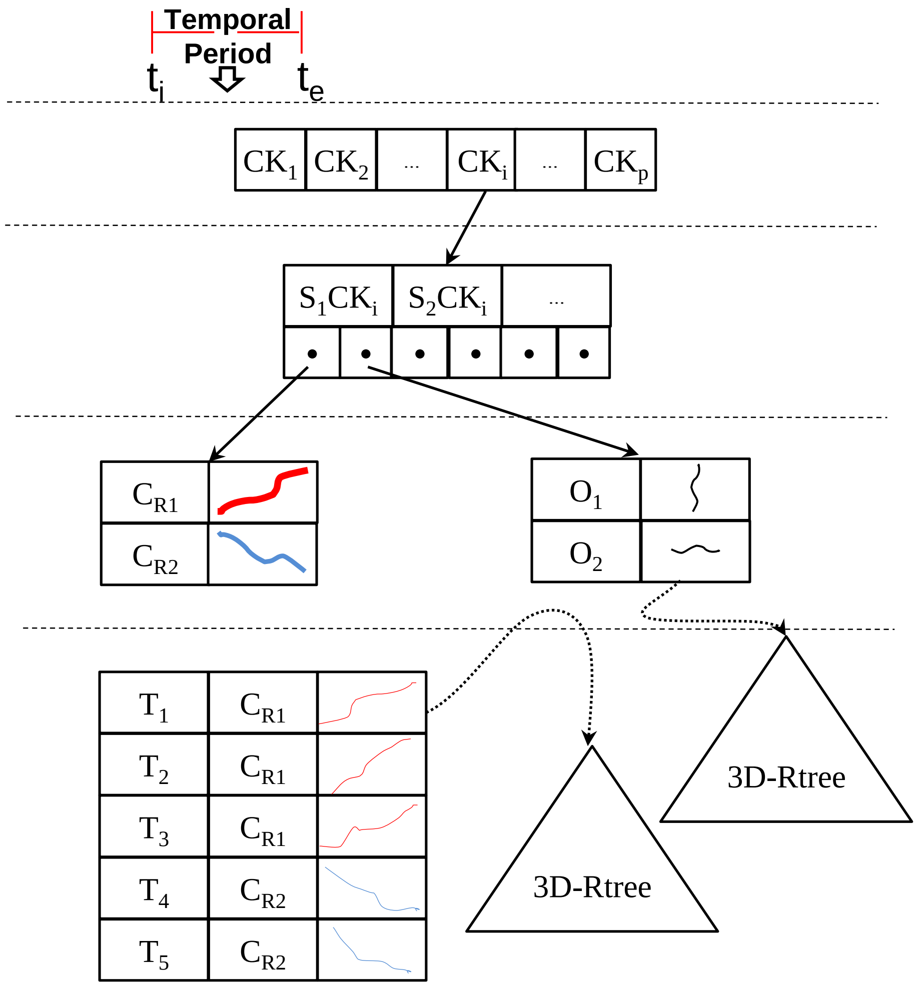
- 3^{rd} level S^2T-Clustering
  - For each of the resulting new outlier sub-trajectories we re-insert the sub-trajectory from the top of the ReTraTree structure.



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

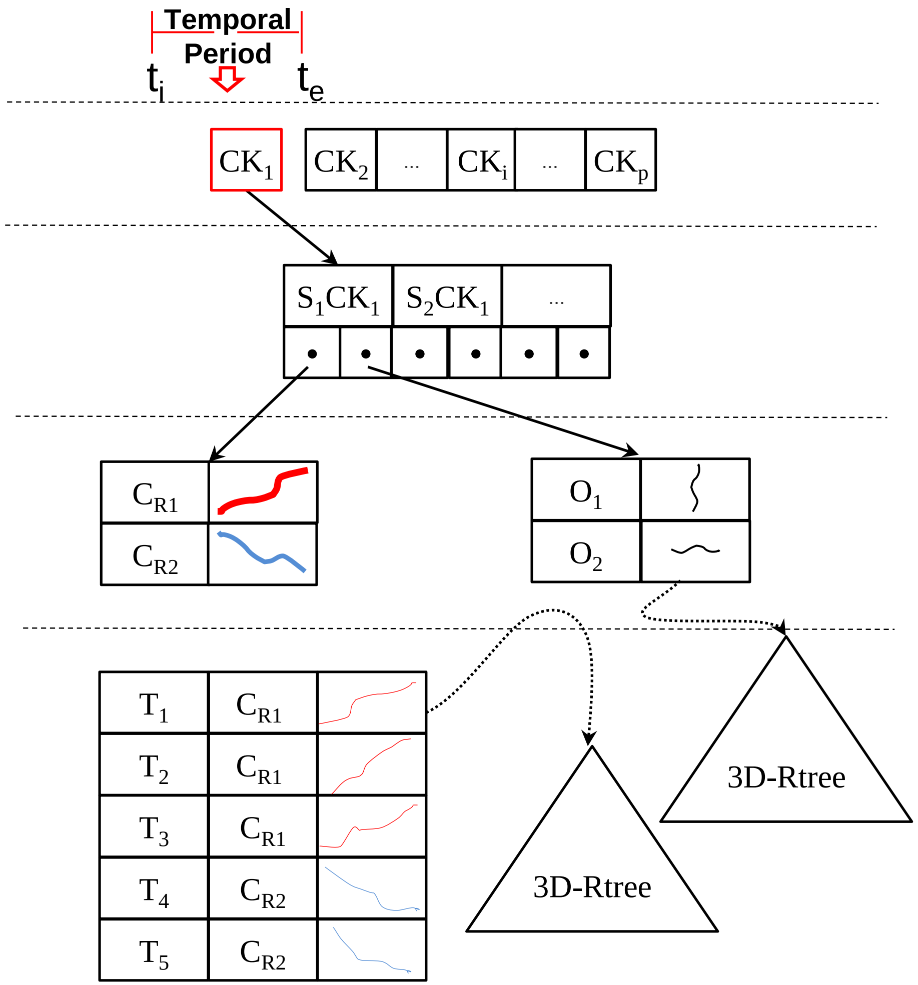
- Input: a temporal period.
- Output: all maximal (w.r.t. time dimension) clusters during the given period of time.



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

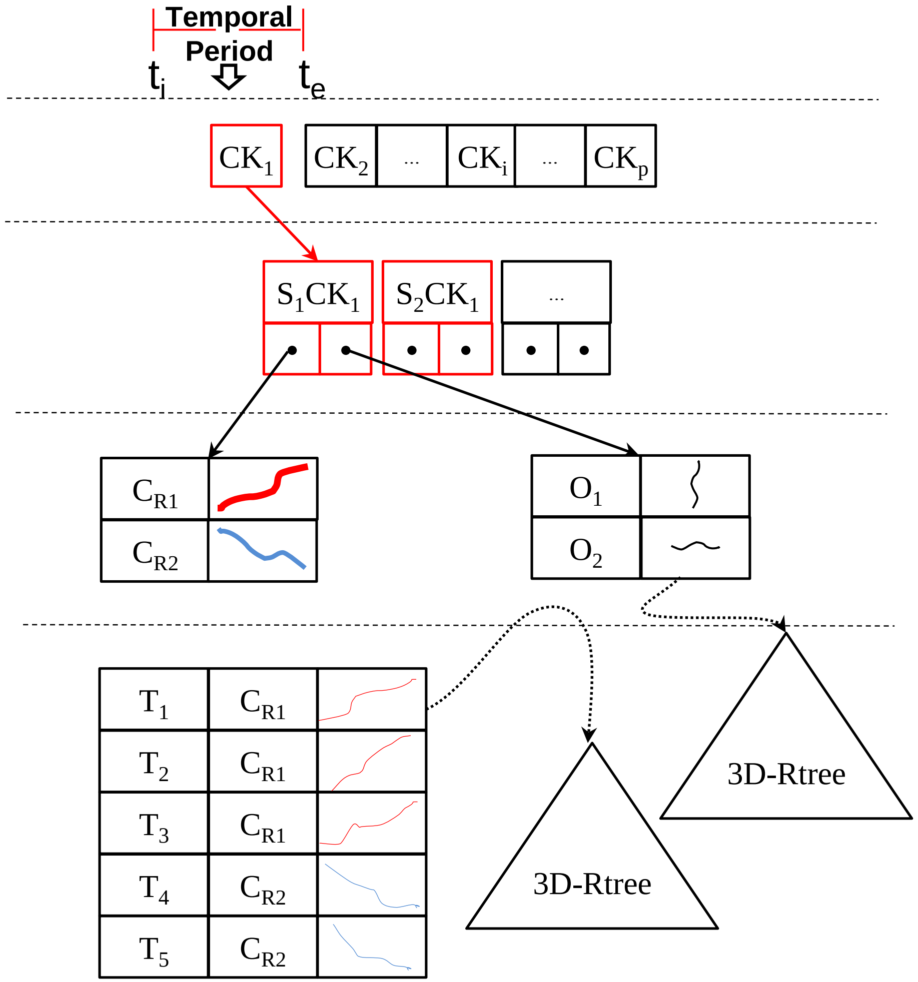
1. Filter the chunks that overlap the given period and for each of them filter the corresponding valid sub-chunks.



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

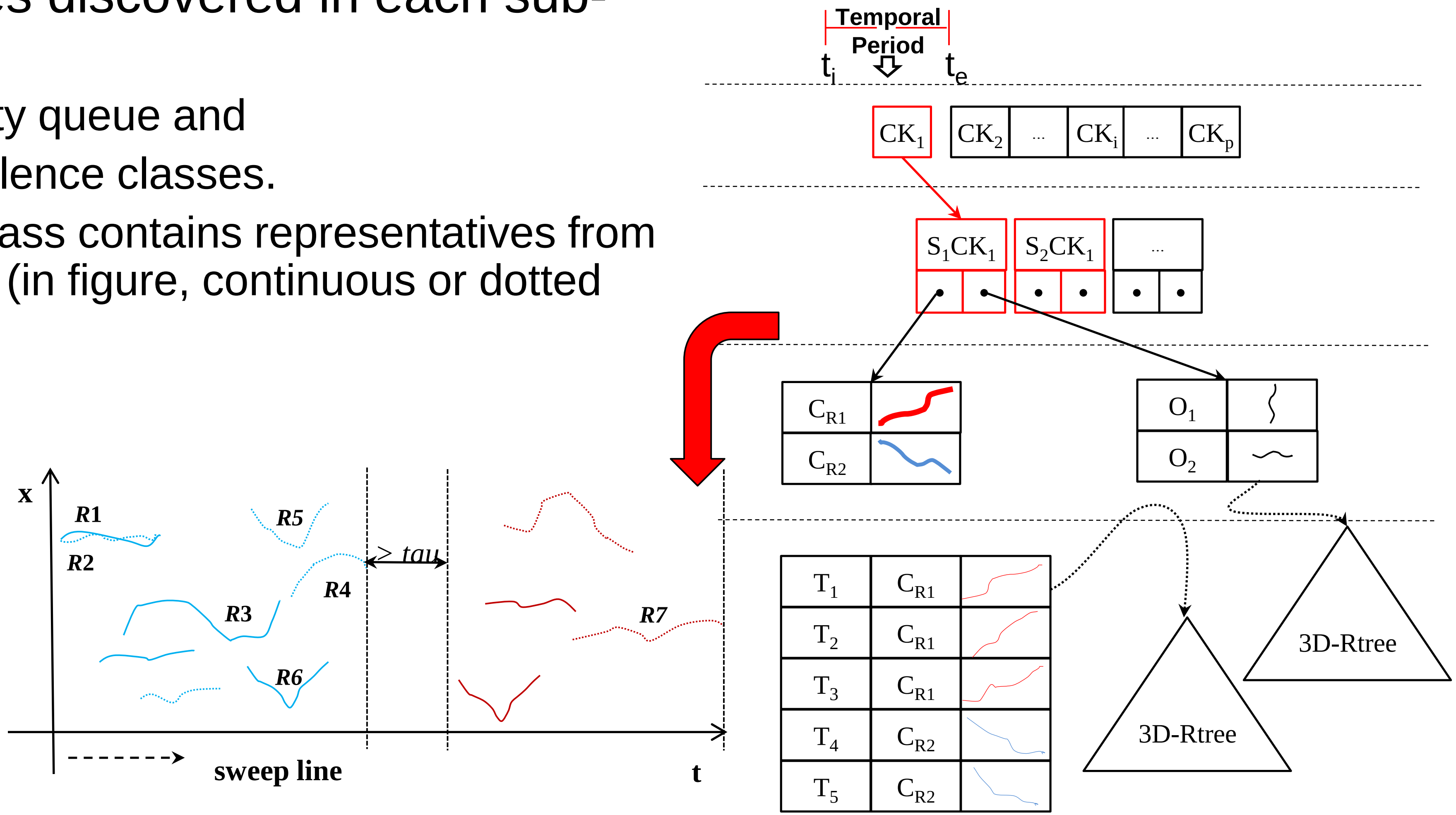
1. Filter the chunks that overlap the given period and for each of them filter the corresponding valid sub-chunks.



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

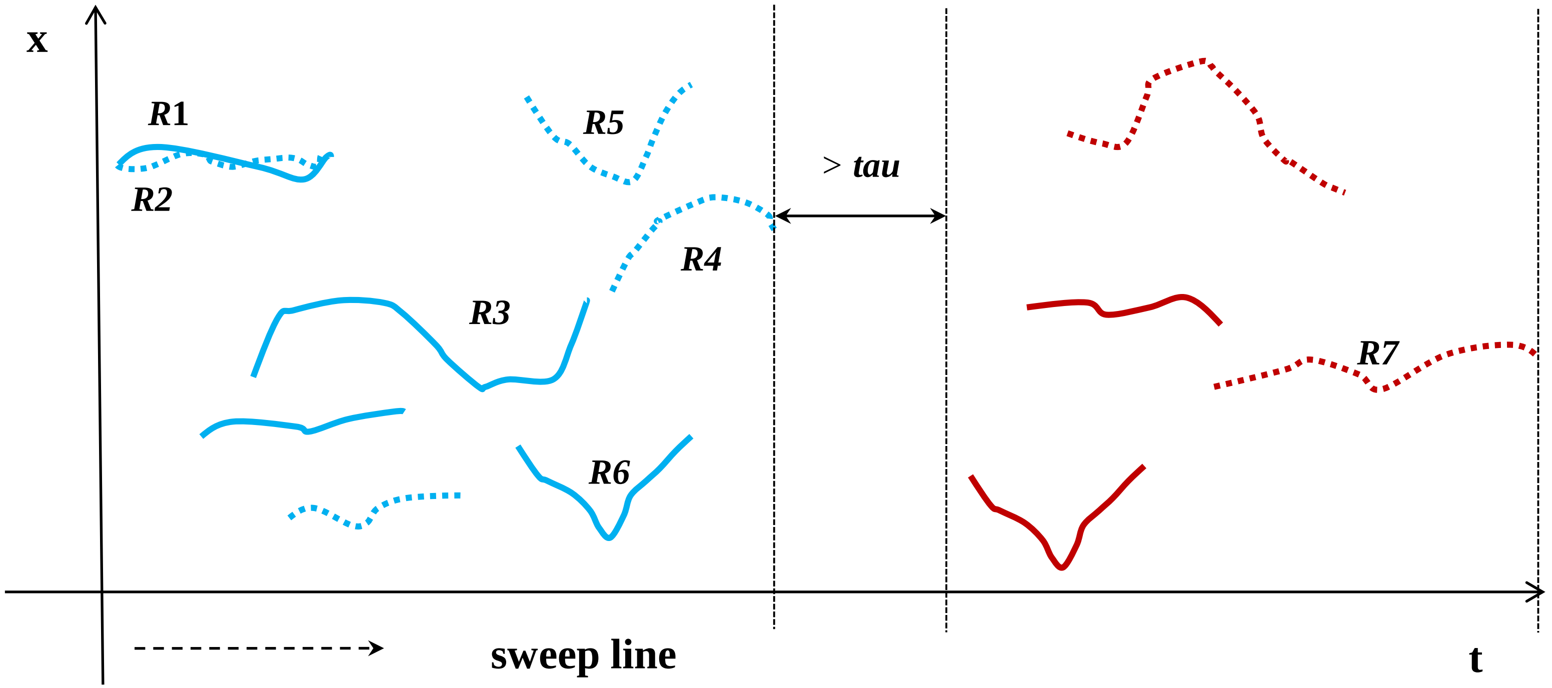
2. The representatives discovered in each sub-chunk are
   - organized in a priority queue and
   - partitioned in equivalence classes.
   - Each equivalence class contains representatives from multiple sub-chunks (in figure, continuous or dotted lines).



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

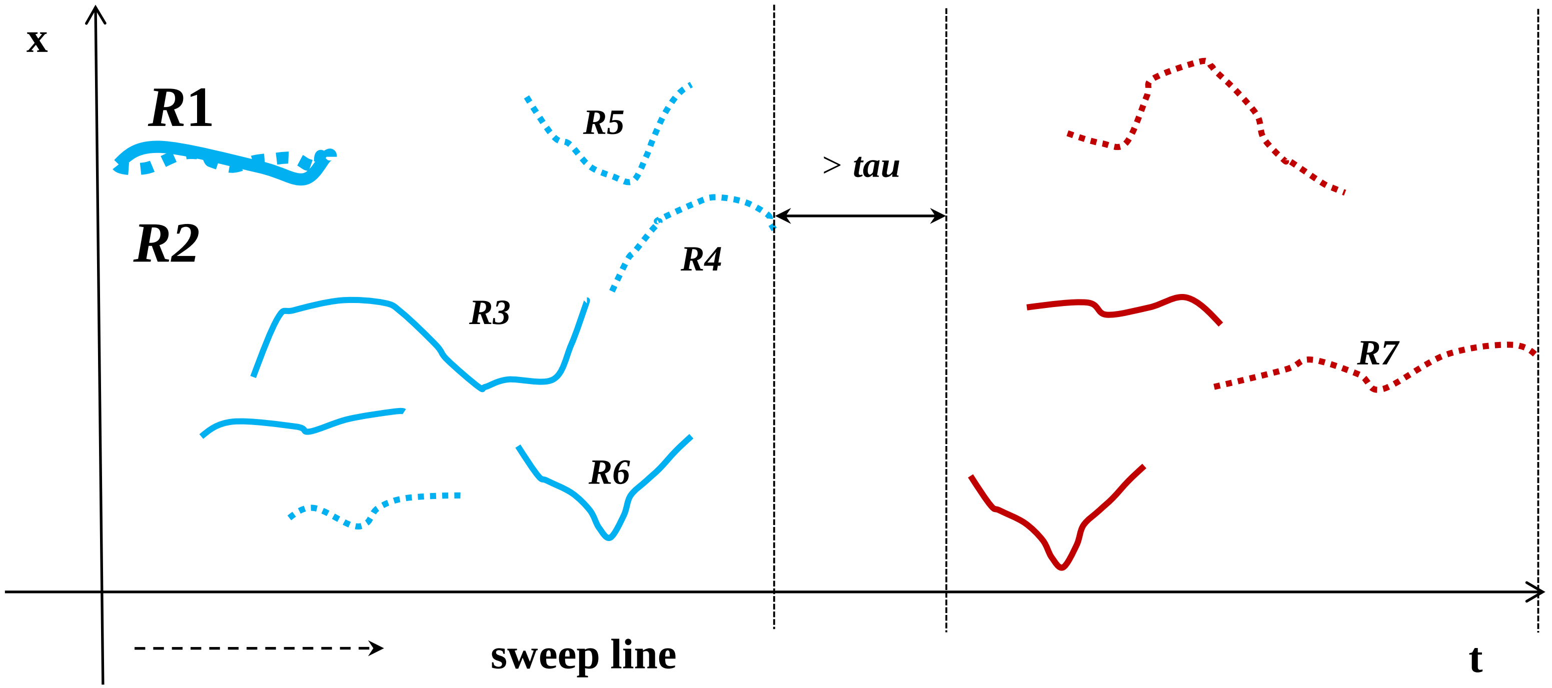
3. For each equivalence class, **sweep** through the time dimension



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

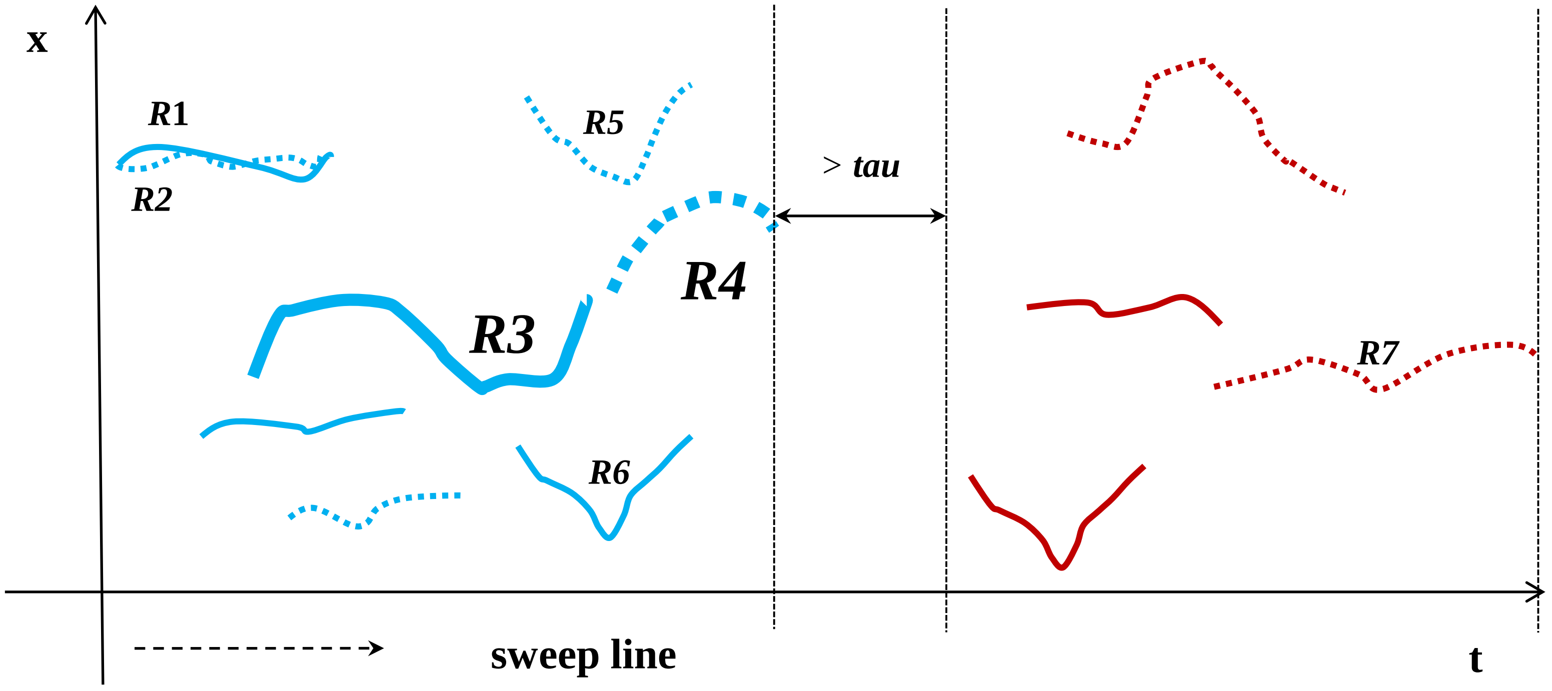
4. **Merge** similar subtrajectories from different subchunks



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

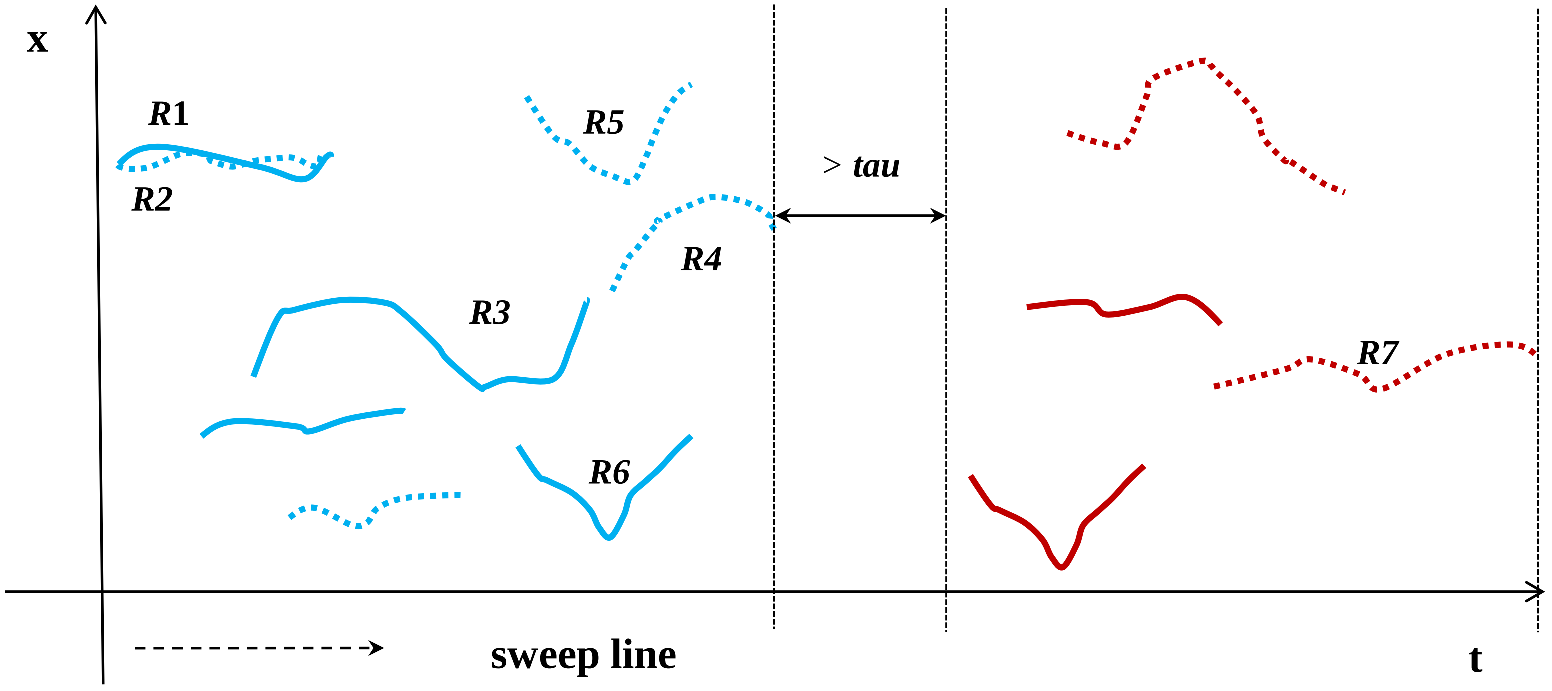
5. **Append** subtrajectories in order to get maximal patterns



Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

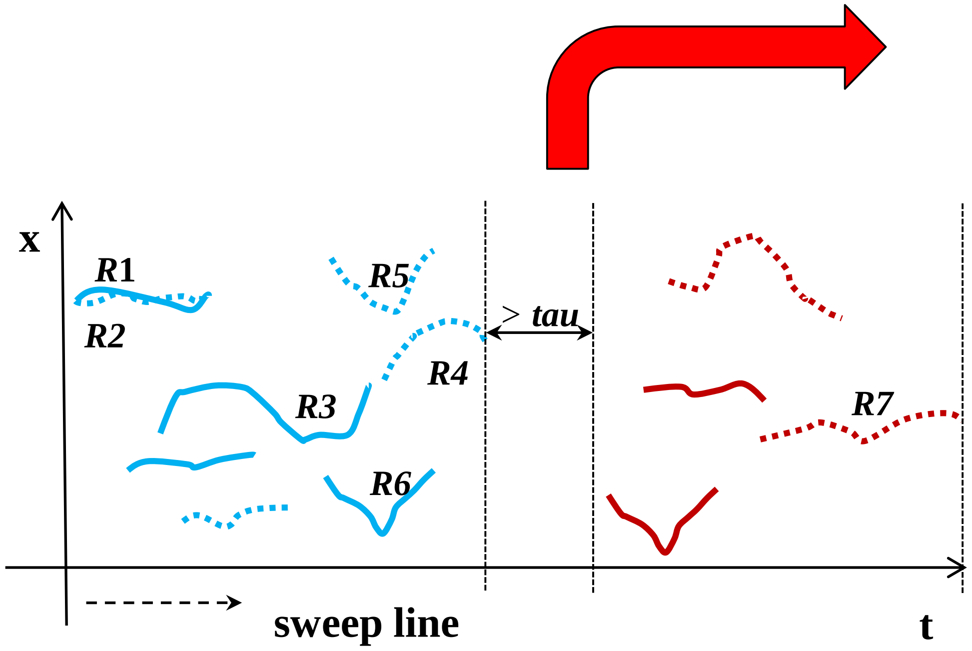
6. In any other case the algorithm does nothing, (continues to the next pair).

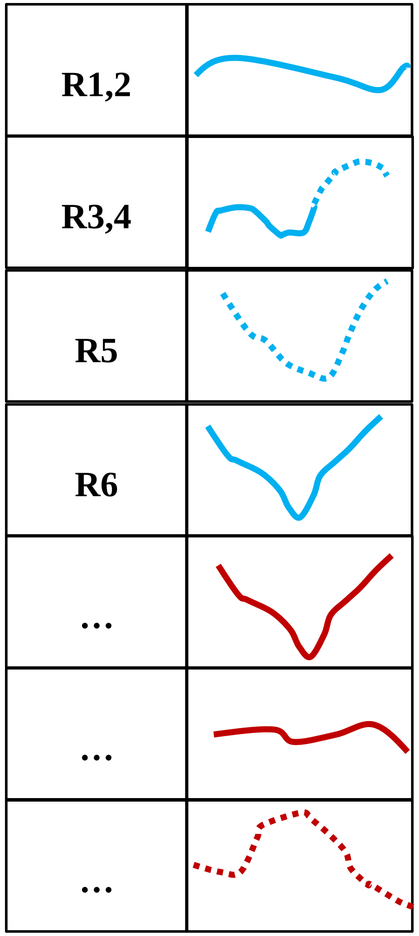


Temporal-constrained Subtrajectory Cluster Analysis

ReTraTree in Action - QuT-Clustering

7. The output representatives are:
   - the **merged** representatives,
   - the **appended** representatives and the
   - **rest** of the representatives.

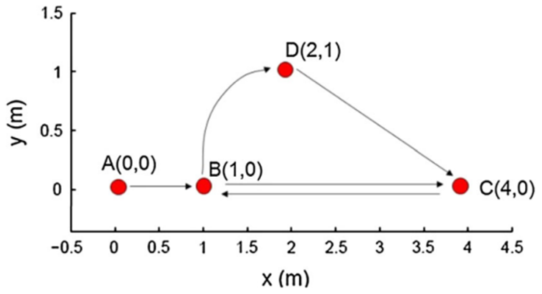


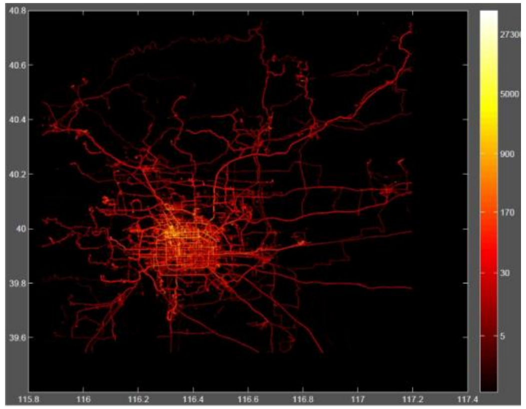


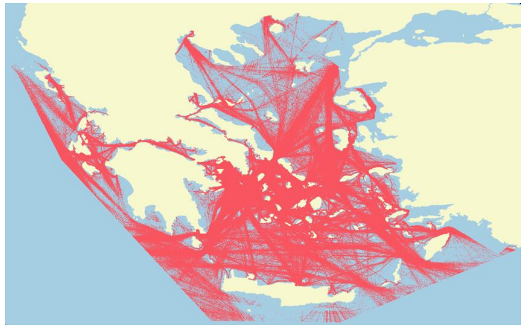
Temporal-constrained Subtrajectory Cluster Analysis

Experimental Study – Datasets

Statistic	SMOD	GeoLife	$IMIS_2$
# Trajectories	400	18668	5110
# Segments	35273	24159325	443657
Dataset Duration (hh:mm:ss)	0:02:00	1932 days 22:59:48	6 days 19:59:53
Avg. Sampling Rate (hh:mm:ss)	0:00:01	0:00:08	0:18:02
Avg. Segment Length (m)	8	72	1545
Avg. Segment Speed (m/s)	7.83	5.01	7.03
Avg. Trajectory Speed (m/s)	2.86	3.91	4.52
Avg. # Points per Trajectory	89	1295	88
Avg. Trajectory Duration (hh:mm:ss)	0:01:28	2:43:15	11:33:45
Avg. Trajectory Length (m)	691	93046	134,148

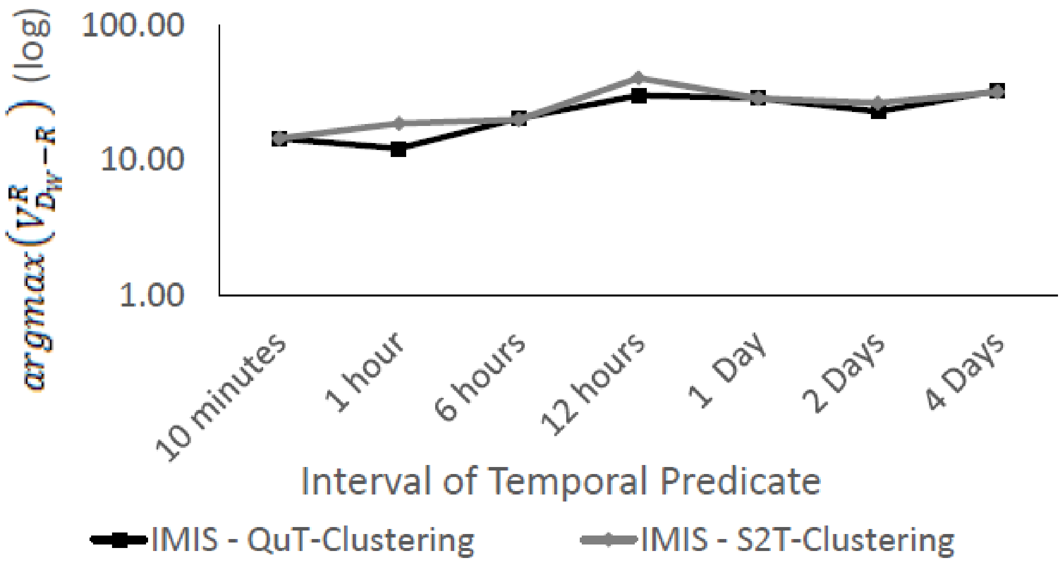


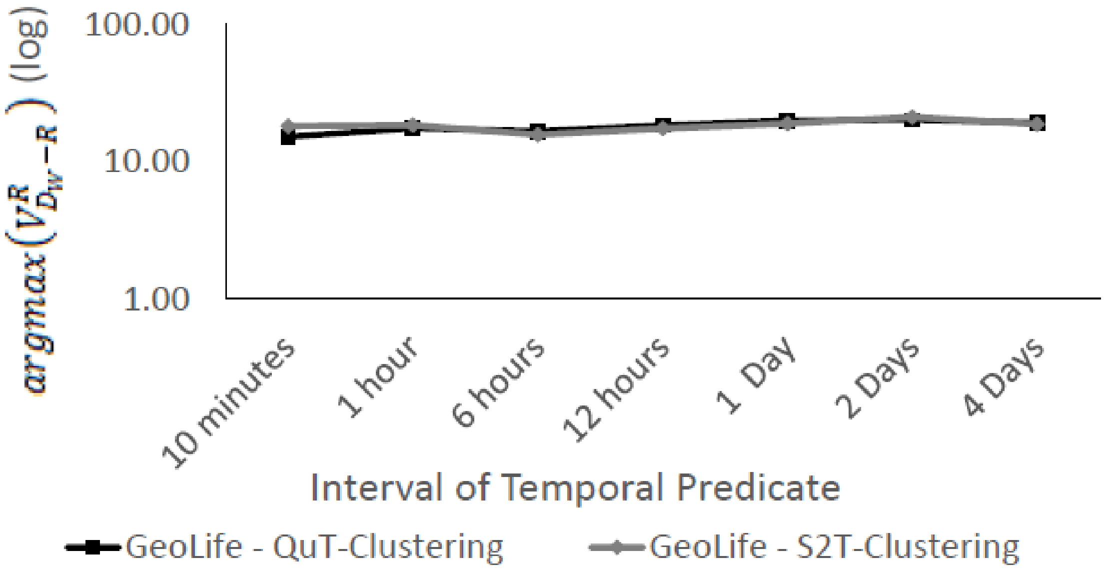




Temporal-constrained Subtrajectory Cluster Analysis

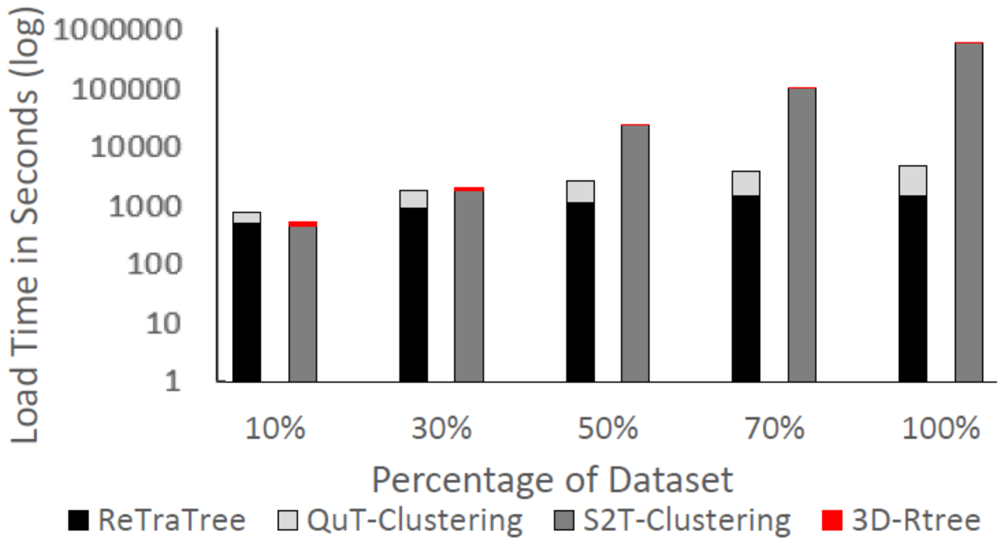
Experimental Study – Quality of Clustering Analysis

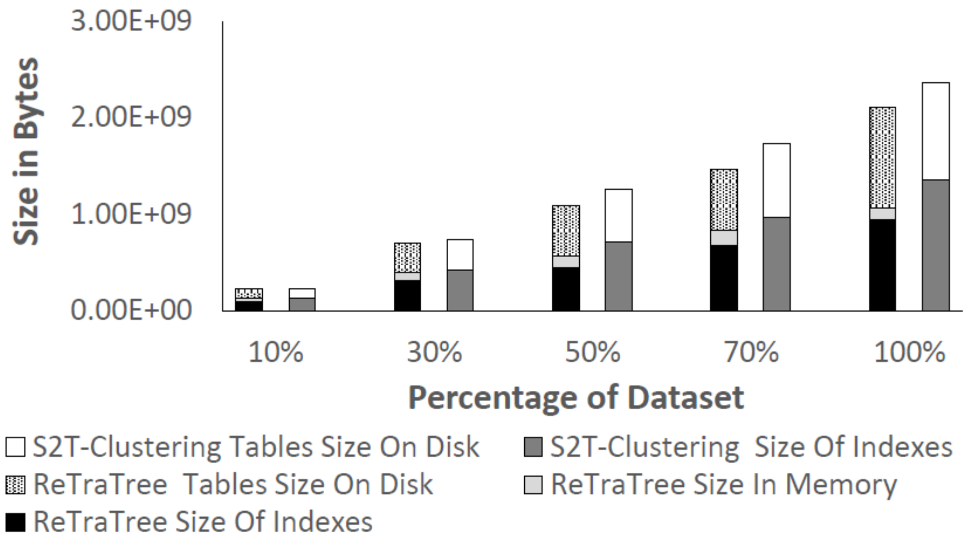




Temporal-constrained Subtrajectory Cluster Analysis

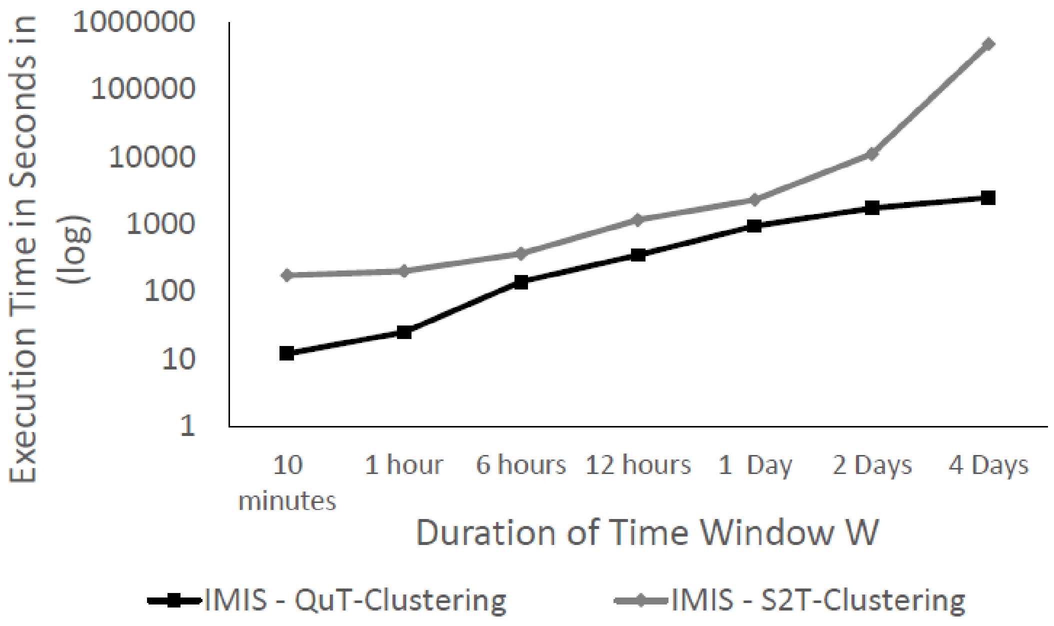
Experimental Study – ReTraTree Maintenance

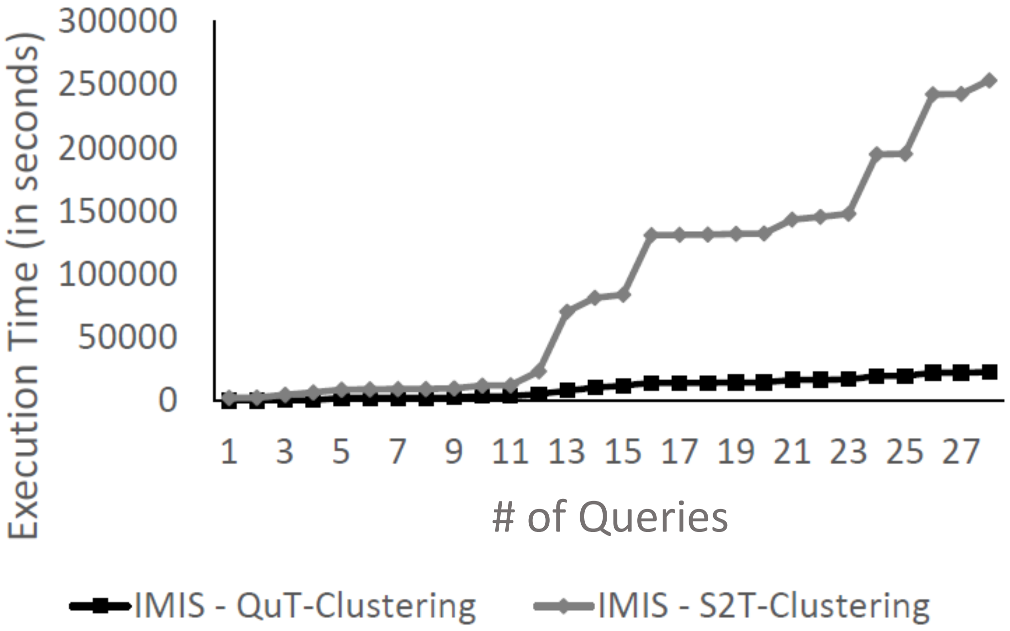




Temporal-constrained Subtrajectory Cluster Analysis

Experimental Study – Efficiency of QuT-clustering vs. S^2T-Clustering





Temporal-constrained Subtrajectory Cluster Analysis

Summary

- We proposed *ReTraTree*, an indexing scheme which organizes trajectories by using an effective spatio-temporal partitioning technique.
- We devised *QuT-Clustering*, a query operator on top of *ReTraTree*, to solve the problem of temporal-constrained subtrajectory clustering.
- Our approach outperforms S^2T-Clustering, the state-of-the-art in-DBMS solution supported by PostgreSQL, by several orders of magnitude without compromising the quality of the results.

Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

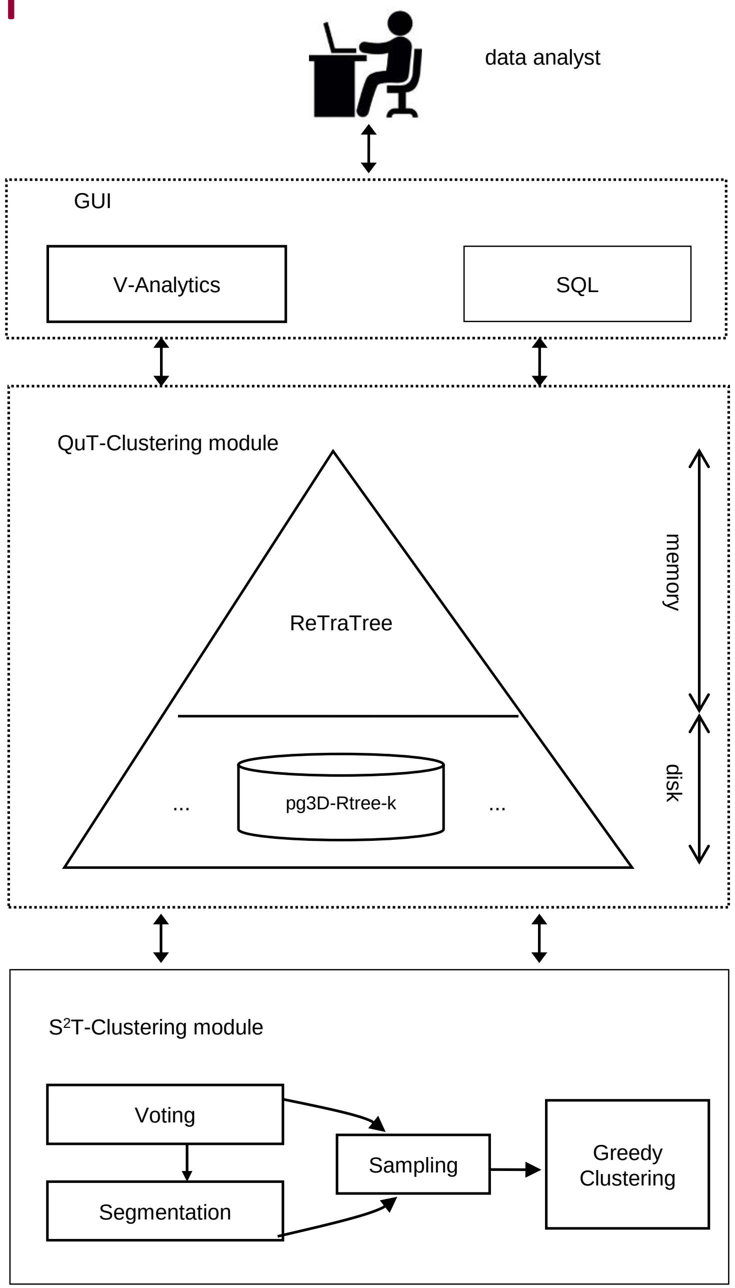
Introduction

- The goal:
  - **Interactive mobility data exploration** and **analysis** and more specifically, **progressive time-aware sub-trajectory cluster analysis.**
- Desired specifications
  - **Efficiency; Ease of use** (via an SQL interface); **Interactive Visual Analytics** support
- Our proposal
  - Implement two state of the art **efficient** and **scalable** solutions for **subtrajectory clustering** [12][13] that are incorporated in **Hermes@PostgreSQL** [14], a **MOD** which is built on top of a **real-world DBMS**.
  - Integrate with a **Visual Analytics** tool (V-Analytics) to facilitate real world **interactive analysis**.

Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

Major Modules and System Architecture

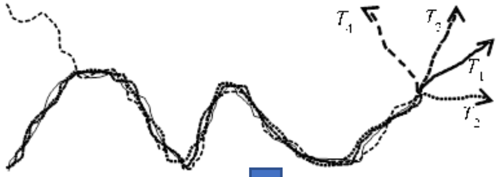
- V-Analytics
- QuT-Clustering
- ReTraTree
- S²T-Clustering



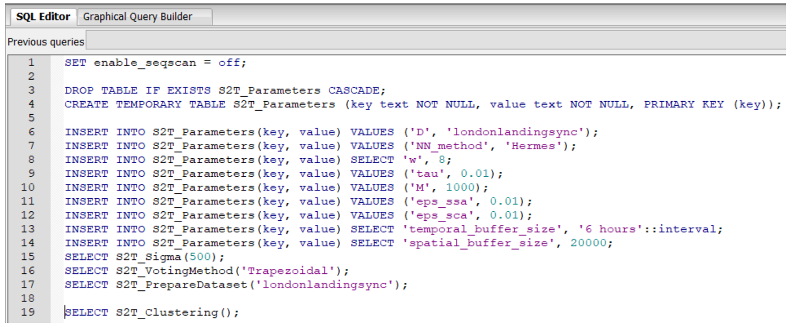
Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

S^{2}T-Clustering Module

Input:



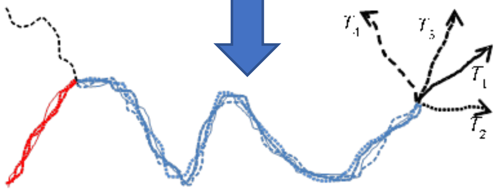
SQL Interface of S^{2}T-Clustering:

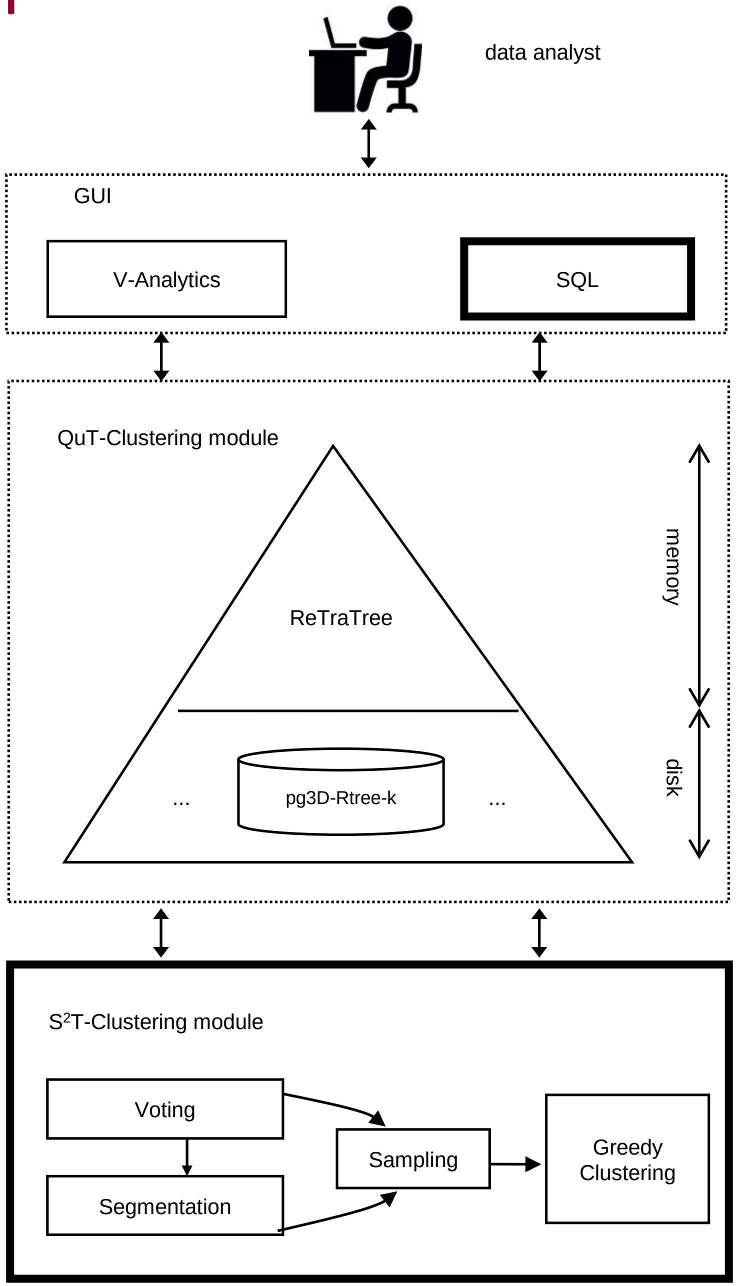

SQL Editor | Graphical Query Builder

Previous queries

```sql
SET enable_seqscan = off;

DROP TABLE IF EXISTS S2T_Parameters CASCADE;
CREATE TEMPORARY TABLE S2T_Parameters (key text NOT NULL, value text NOT NULL, PRIMARY KEY (key));

INSERT INTO S2T_Parameters(key, value) VALUES ('D', 'londonlandingsync');
INSERT INTO S2T_Parameters(key, value) VALUES ('NN_method', 'Hermes');
INSERT INTO S2T_Parameters(key, value) SELECT 'w', 8;
INSERT INTO S2T_Parameters(key, value) VALUES ('tau', 0.01);
INSERT INTO S2T_Parameters(key, value) VALUES ('M', 1000);
INSERT INTO S2T_Parameters(key, value) VALUES ('eps_ssa', 0.01);
INSERT INTO S2T_Parameters(key, value) VALUES ('eps_sca', 0.01);
INSERT INTO S2T_Parameters(key, value) SELECT 'temporal_buffer_size', '6 hours'::interval;
INSERT INTO S2T_Parameters(key, value) SELECT 'spatial_buffer_size', 20000;
SELECT S2T_Sigma(500);
SELECT S2T_VotingMethod('Trapezoidal');
SELECT S2T_PrepareDataset('londonlandingsync');

SELECT S2T_Clustering();
```

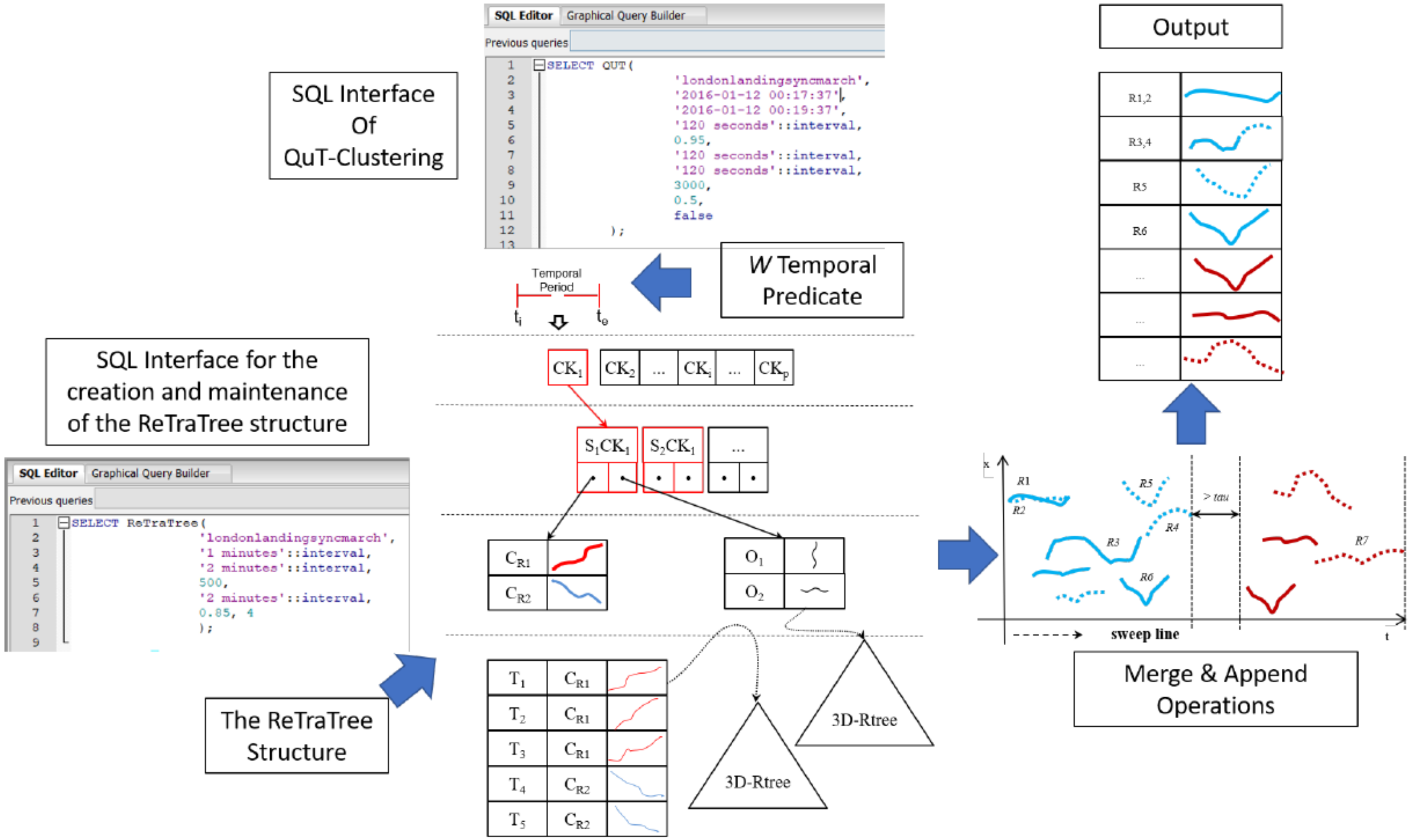
Output:

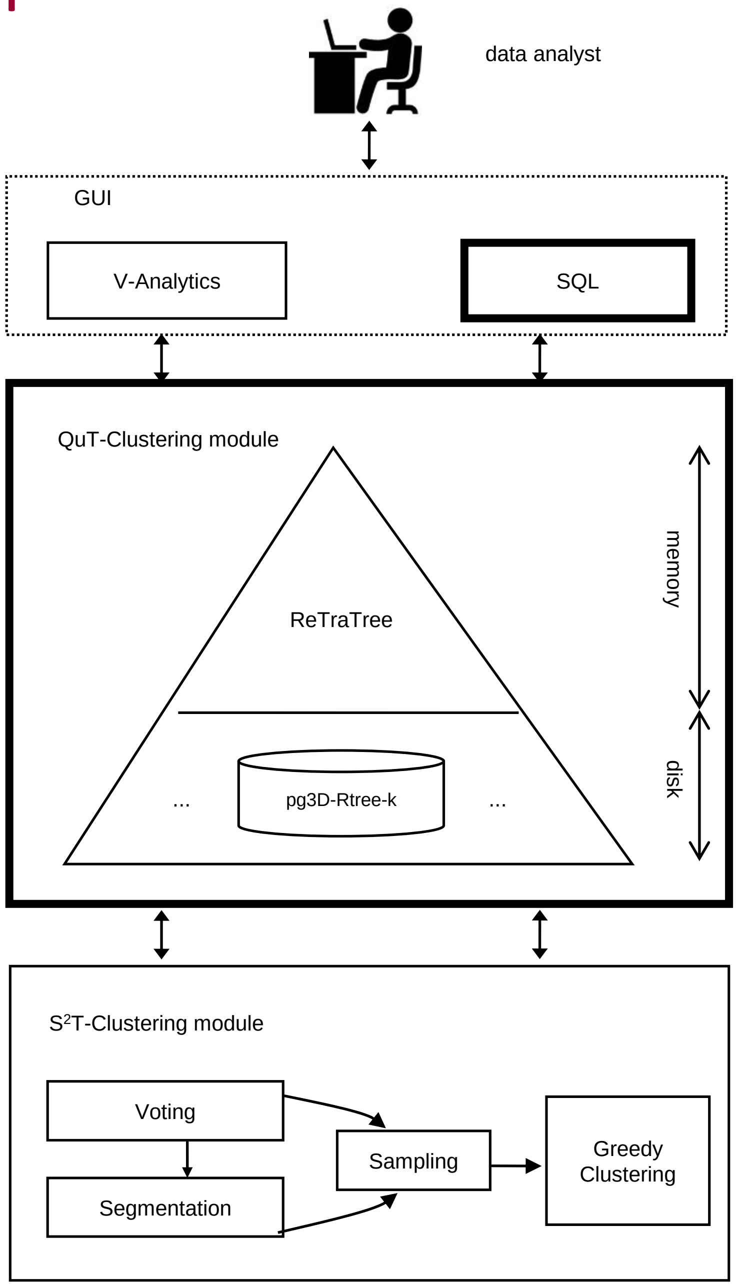




Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

QuT-Clustering Module

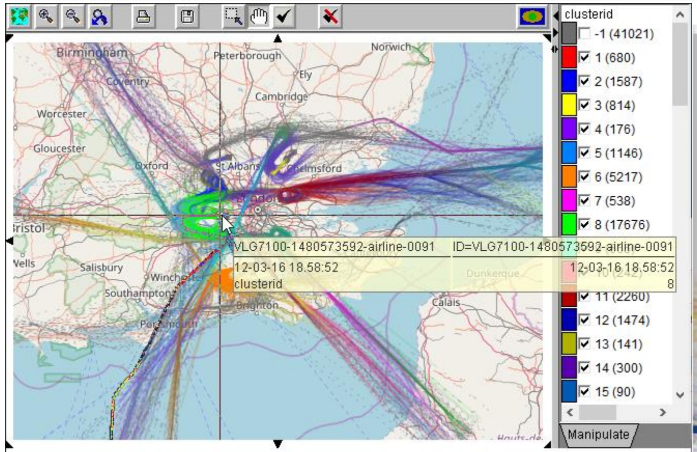




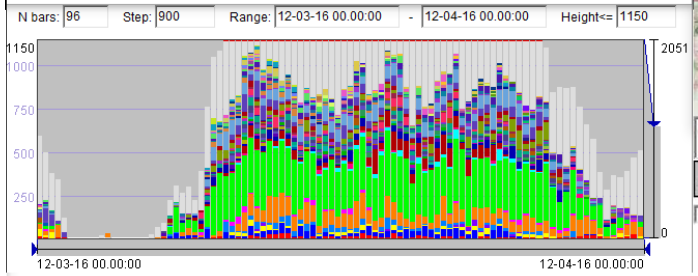
Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

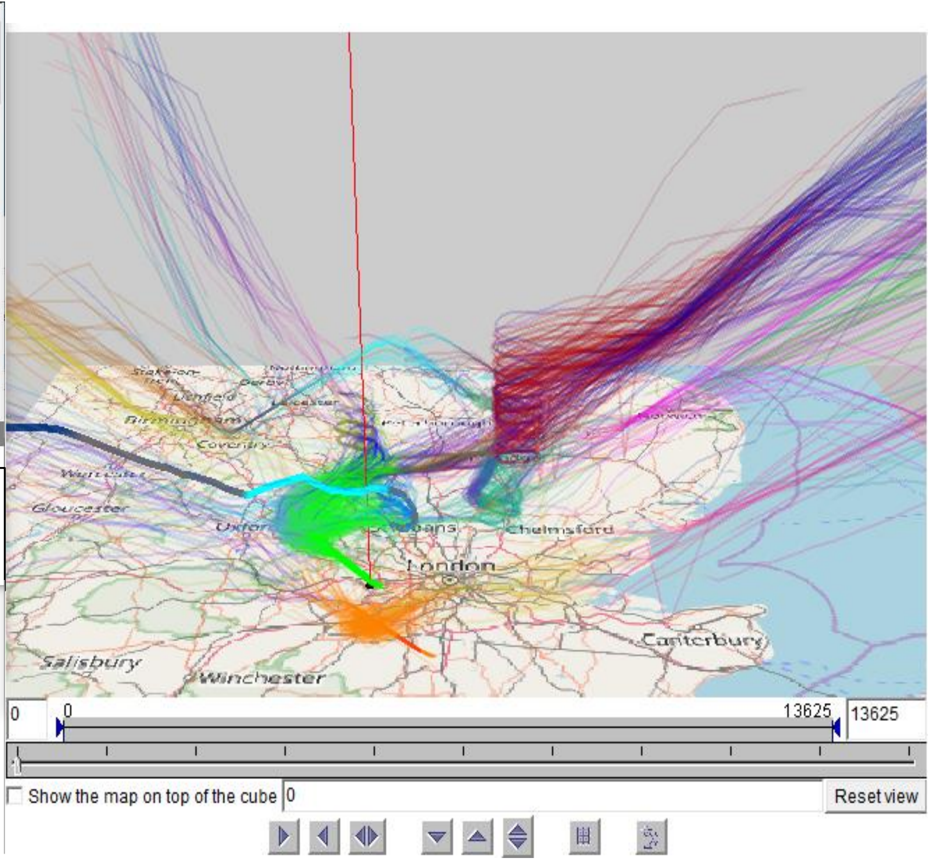
V-Analytics Module

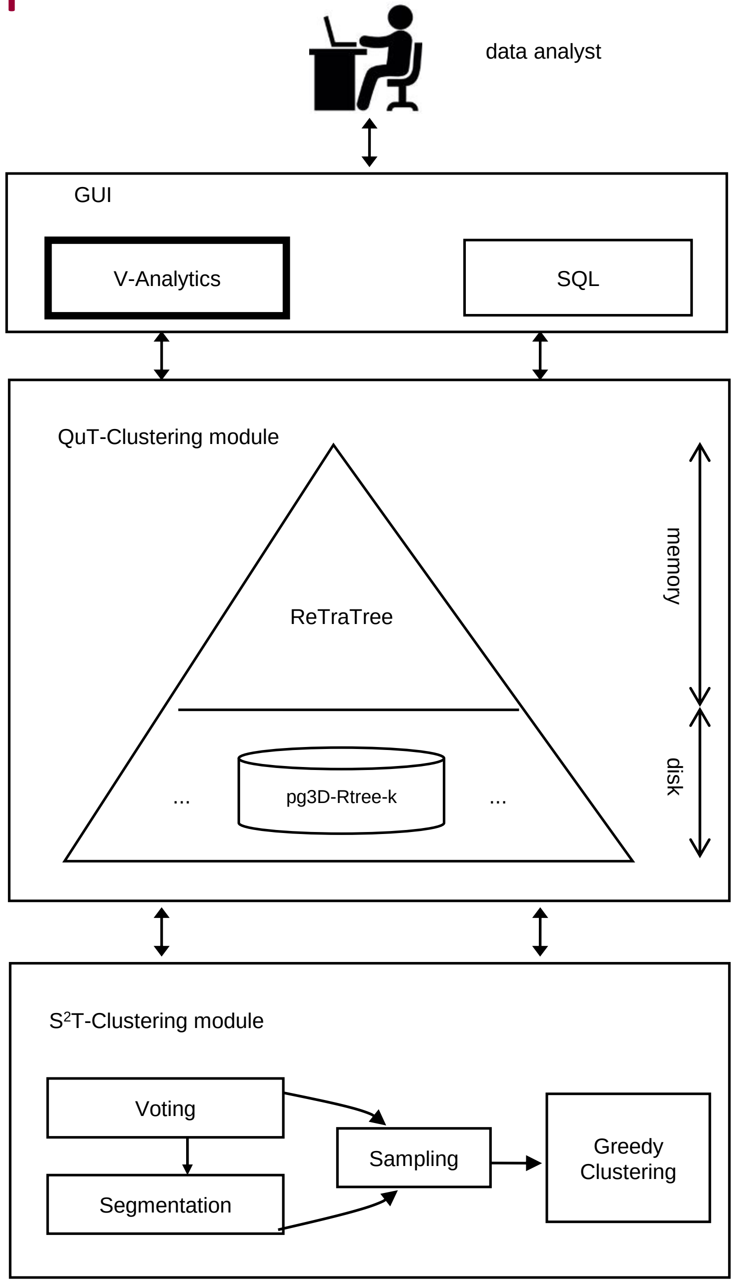
Map display of clusters	3D shapes of cluster members



Evolution of cardinality of clusters over time





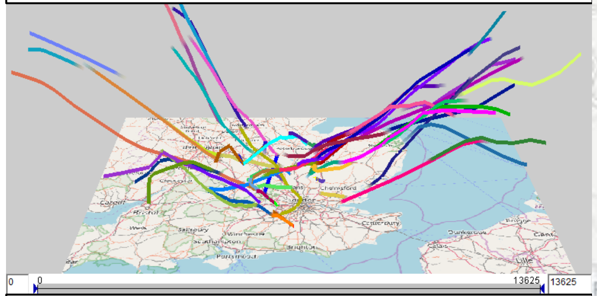


Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

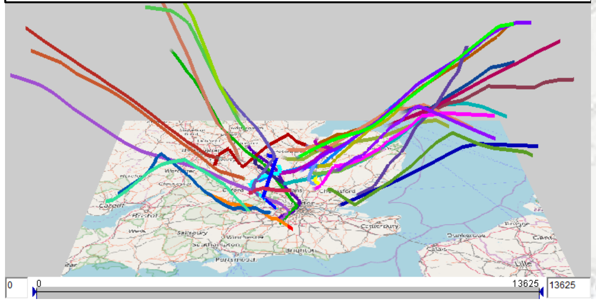
Demonstration of Results

Comparison of 2 runs of S²T-Clustering

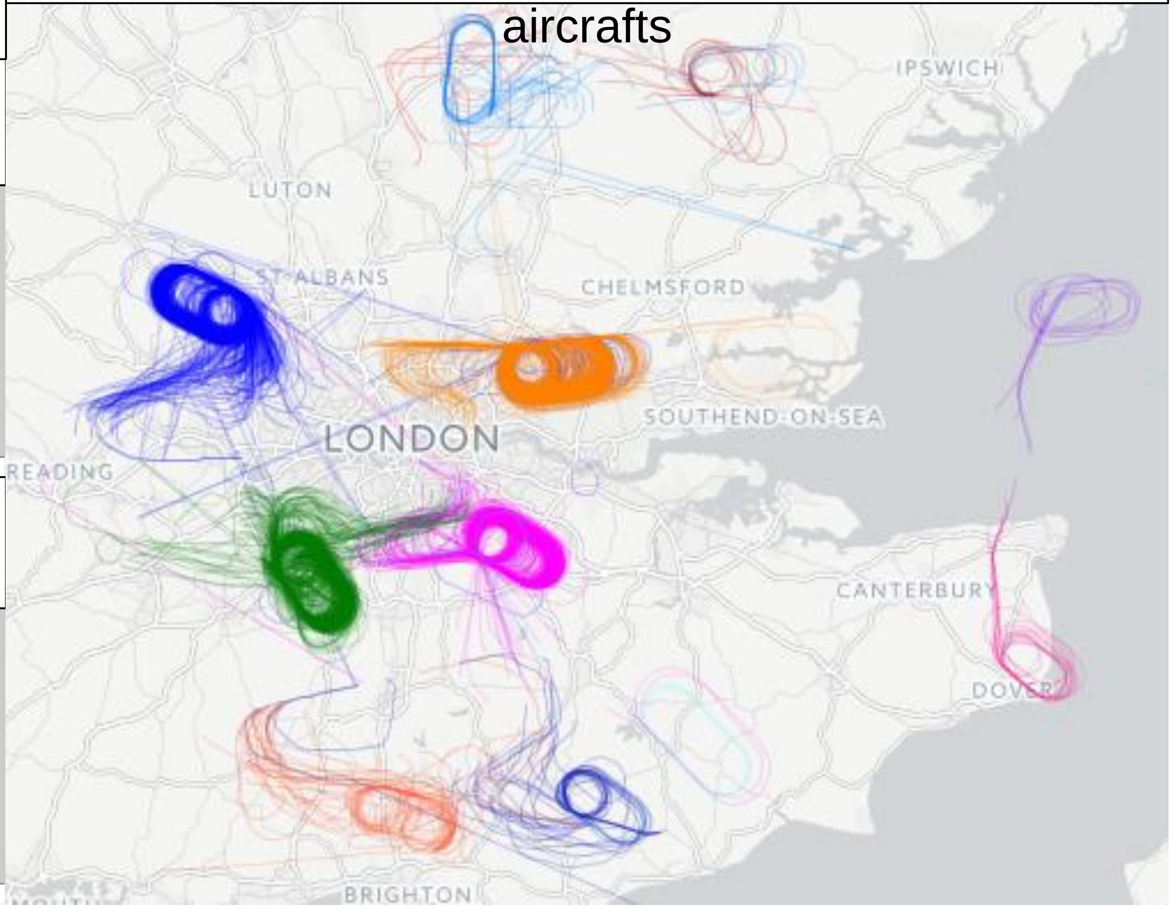
Run 1: Small Sub-Trajectories



Run 2: Large Sub-Trajectories



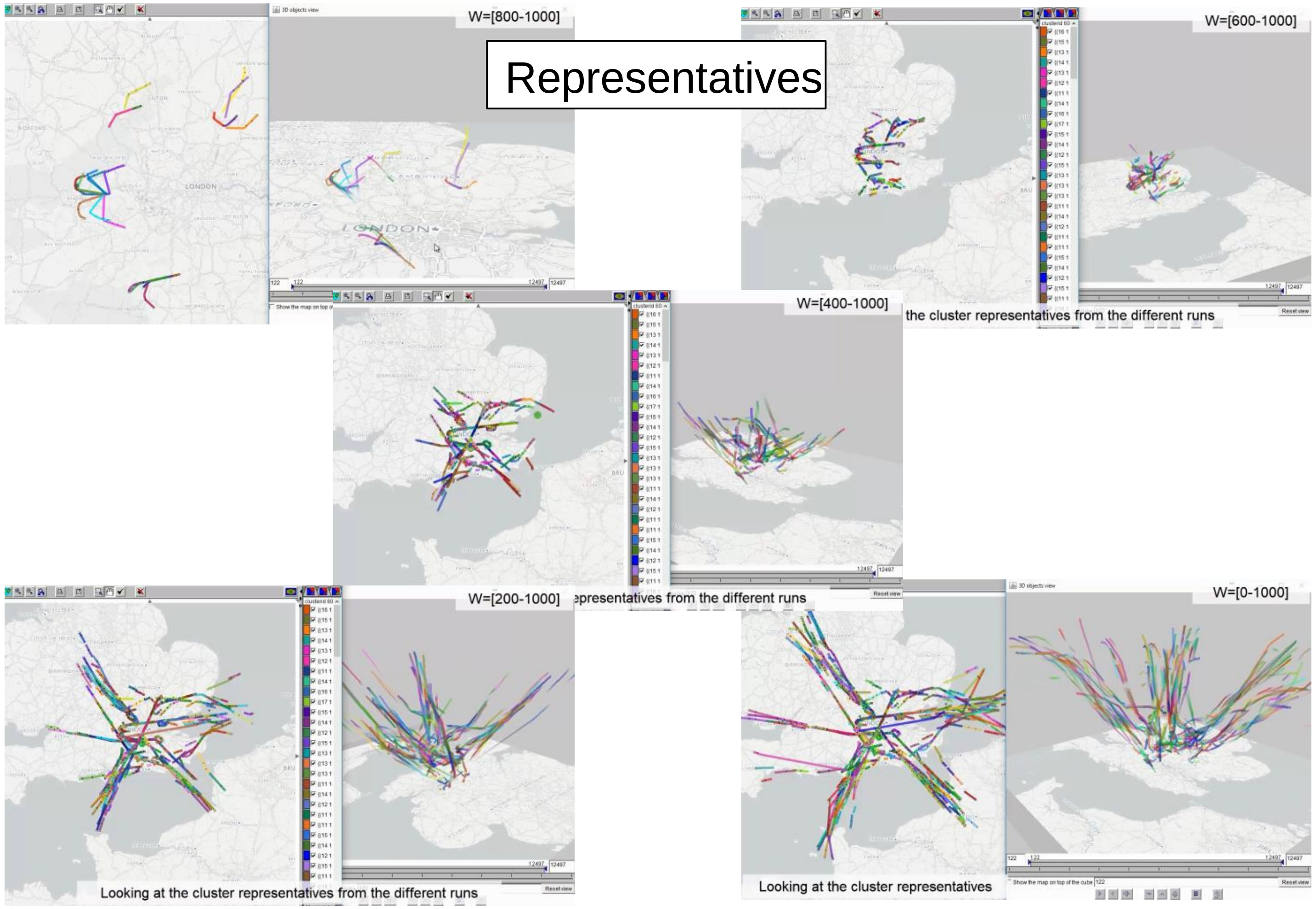
Discovery of **holding patterns** performed by aircrafts



Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

Demonstration of Results

- Exploration of results of 5 runs of QuT-Clustering.
- Each run has an increasing temporal predicate
  - W: 800-1000, 600-1000, 400-1000, 200-1000, 0-1000
- We ask for patterns that are valid during the specified period.
- The time is specified in relative units from 0 to 1000.



Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL

Summary

- We presented an efficient in-DBMS framework that facilitates progressive time-aware subtrajectory cluster analysis.
  - spatiotemporal subtrajectory clustering
  - on demand index-based time-aware clustering
- The framework is also extended with a VA tool to facilitate real world analysis.

Publications related to Part I

Originating from this PhD Thesis

1. N. Pelekis, P. Tampakis, M. Vodas, C. Panagiotakis, Y. Theodoridis. **In-DBMS Sampling-based Sub-trajectory Clustering**, In Proceedings of EDBT Conf., 2017.
2. N. Pelekis, P. Tampakis, M. Vodas, C. Doulkeridis Y. Theodoridis. **On Temporal-Constrained Sub-Trajectory Cluster Analysis**, Data Mining and Knowledge Discovery, 31(5):1294-1330, 2017.
3. P. Tampakis, N. Pelekis, N. V. Andrienko, G. L. Andrienko, G. Fuchs, and Y. Theodoridis. **Time-aware Sub-trajectory Clustering in Hermes@PostgreSQL**. In Proceedings of IEEE ICDE Conf., 2018.

Part III
Distributed Algorithms and Techniques

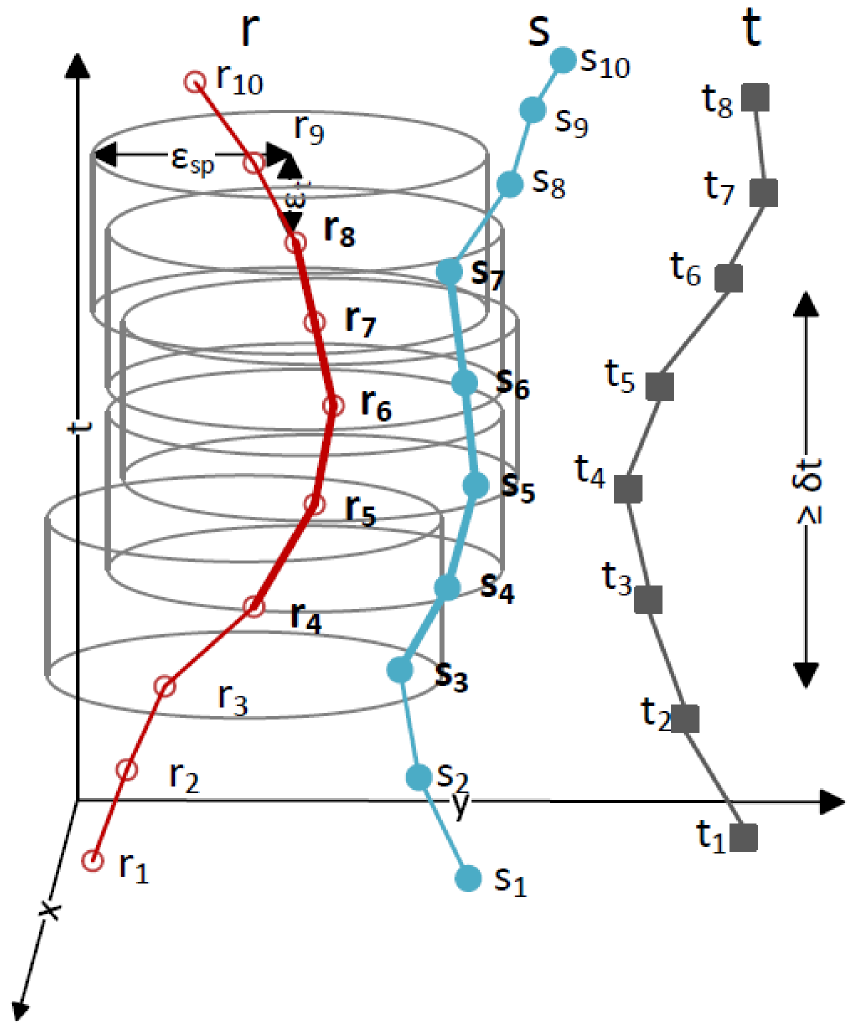
Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Distributed Subtrajectory Join on Massive Datasets

Problem Formulation

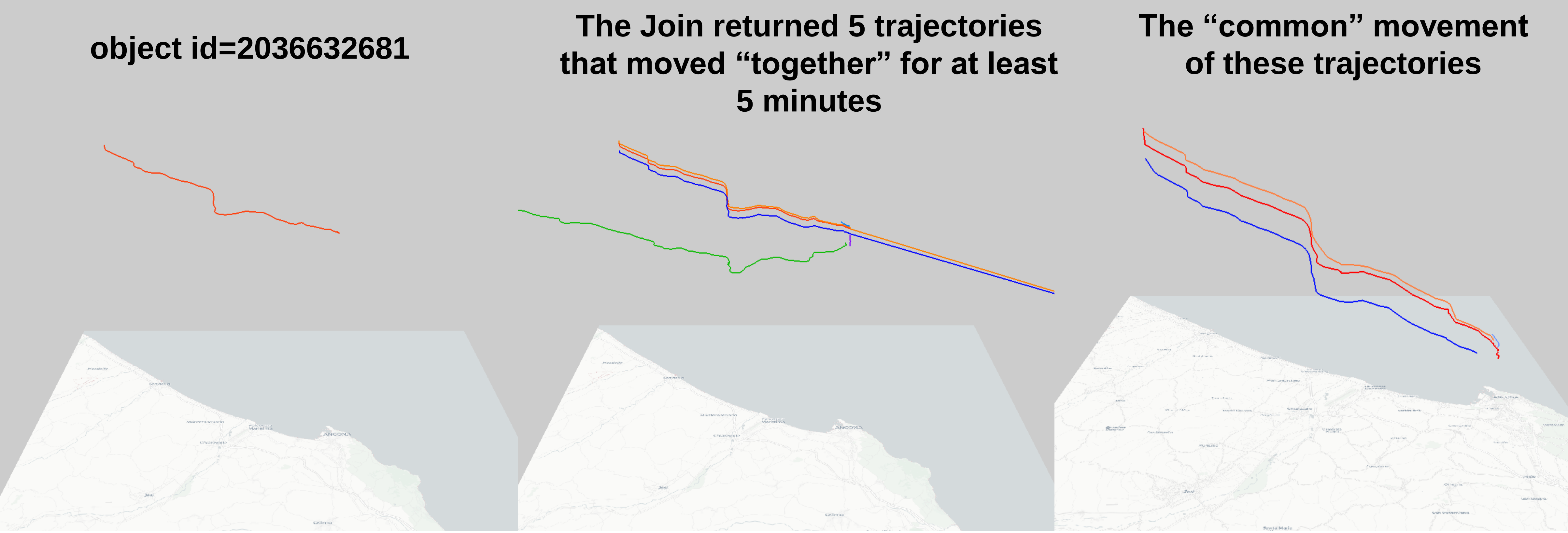
- Given two sets of trajectories, retrieve
  … all pairs of maximal subtrajectories
  … that move “close enough” in time and space
  … for at least some time duration.



Distributed Subtrajectory Join on Massive Datasets

Problem Formulation

- Example



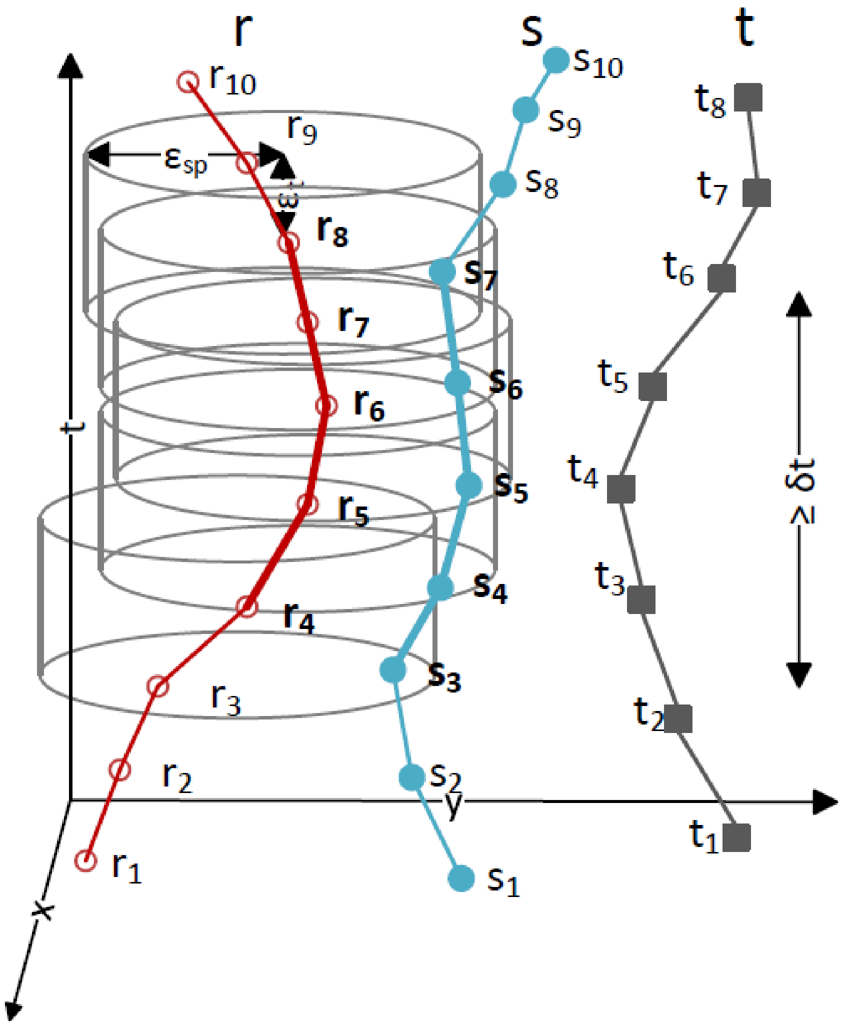
Distributed Subtrajectory Join on Massive Datasets

Problem Formulation

- Given two sets of trajectories, retrieve
  - all pairs of maximally "matching" subtrajectories.

Definition *(Matching subtrajectories) Given a spatial threshold ϵ_{sp}, a temporal tolerance ϵ_t and a time duration δt, a "match" between a pair of subtrajectories (r', s') occurs iff $\Delta w_{r',s'} \geq \delta t - 2\epsilon_t$, and $\forall r'_i \in r'$ there exists at least one $s'_j \in s'$ so that $DistS(r'_i, s'_j) \leq \epsilon_{sp}$ and $DistT(r'_i, s'_j) \leq \epsilon_t$, and $\forall s'_j$ there exists at least one r'_i so that $DistS(s'_j, r'_i) \leq \epsilon_{sp}$ and $DistT(s'_j, r'_i) \leq \epsilon_t$.*

Definition *(Maximally matching subtrajectories) Given a pair of "matching" subtrajectories (r', s') which belong to trajectories r, s respectively, this pair is considered a "maximal match" iff $\nexists$ superset r'' of r' or s'' of s' where the pair (r'', s') or (r', s'') or (r'', s'') would be "matching".*

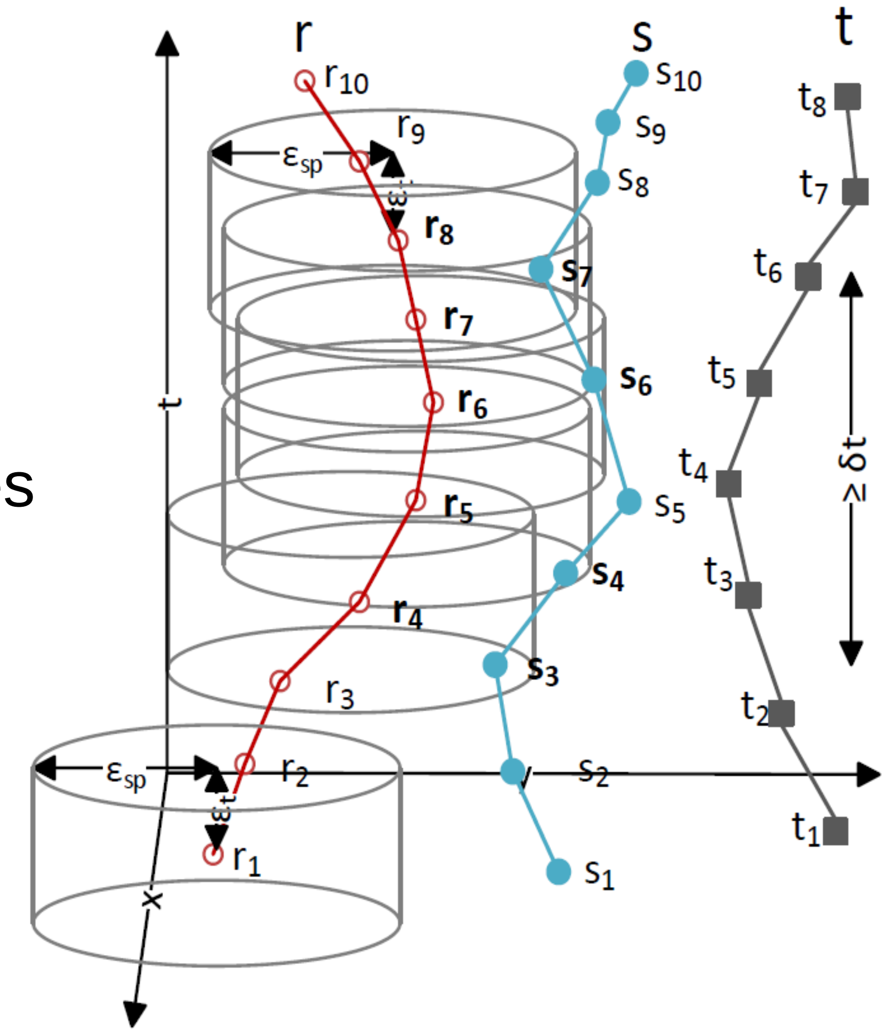


Problem Formulation

Definition *(Joining points) A pair of points (r_i, s_j), where $r_i \in r$ and $s_j \in s$, is a pair of joining points iff they satisfy the following property:*

$$DistS(r_i, s_j) \leq \epsilon_{sp} \text{ and } DistT(r_i, s_j) \leq \epsilon_t.$$

- In fact, the set of ***Joining points*** is the outcome of inner join R⋈S, where the evaluated join predicates are ε_{sp} and ε_t.
- However, these pairs of points do not suffice to return the correct query result.

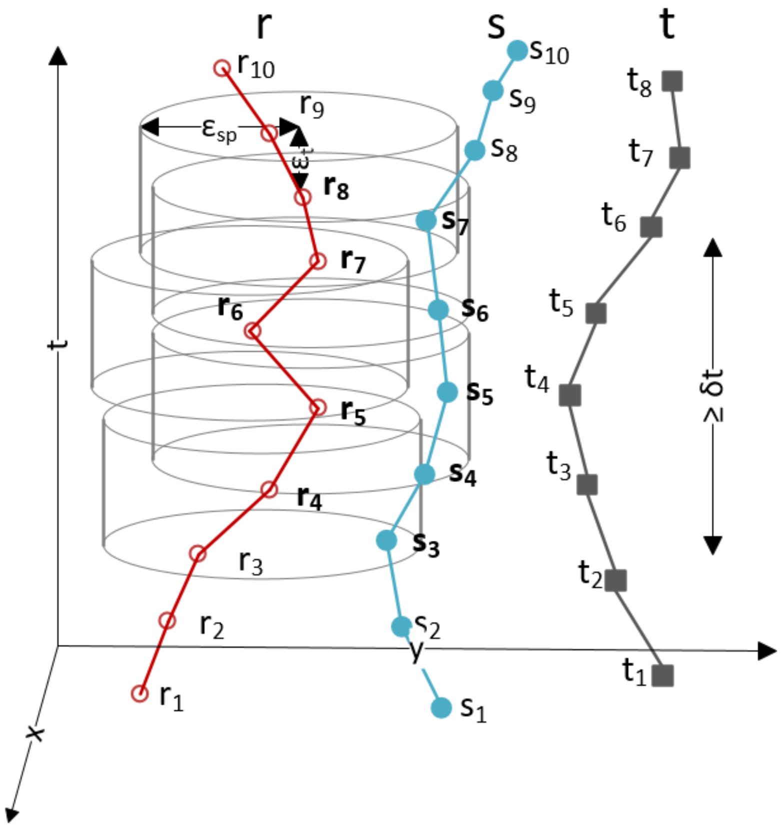


Distributed Subtrajectory Join on Massive Datasets

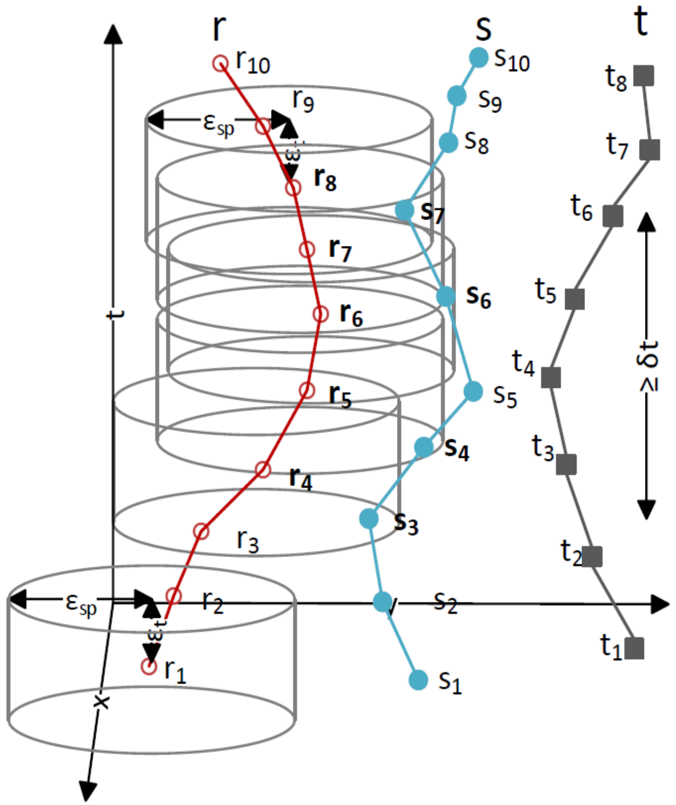
Problem Formulation

- Why *Joining points* are not enough?

Case 1



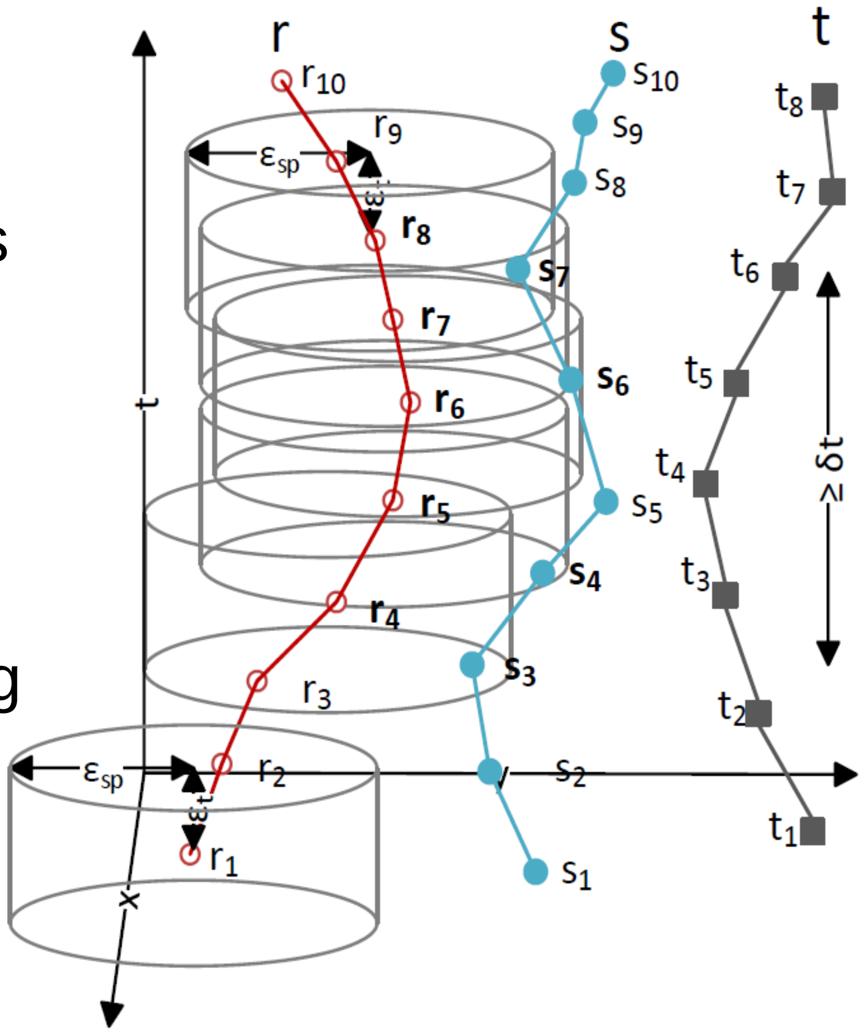
Case 2



Distributed Subtrajectory Join on Massive Datasets

Problem Formulation

- A naïve algorithm would require the Cartesian product *R* × *S* to produce the correct result.
- We claim that *R* × *S* can be represented by two sets of pairs of points, the set of
  - *Joining points* (*JP*) and
  - the set of *non-joining points* (*NJP*).
- Formally, *R* × *S* = *JP* U *NJP*.
- Obviously, trying to deal with our problem by utilizing the Cartesian product is unrealistic.



Distributed Subtrajectory Join on Massive Datasets

Problem Formulation

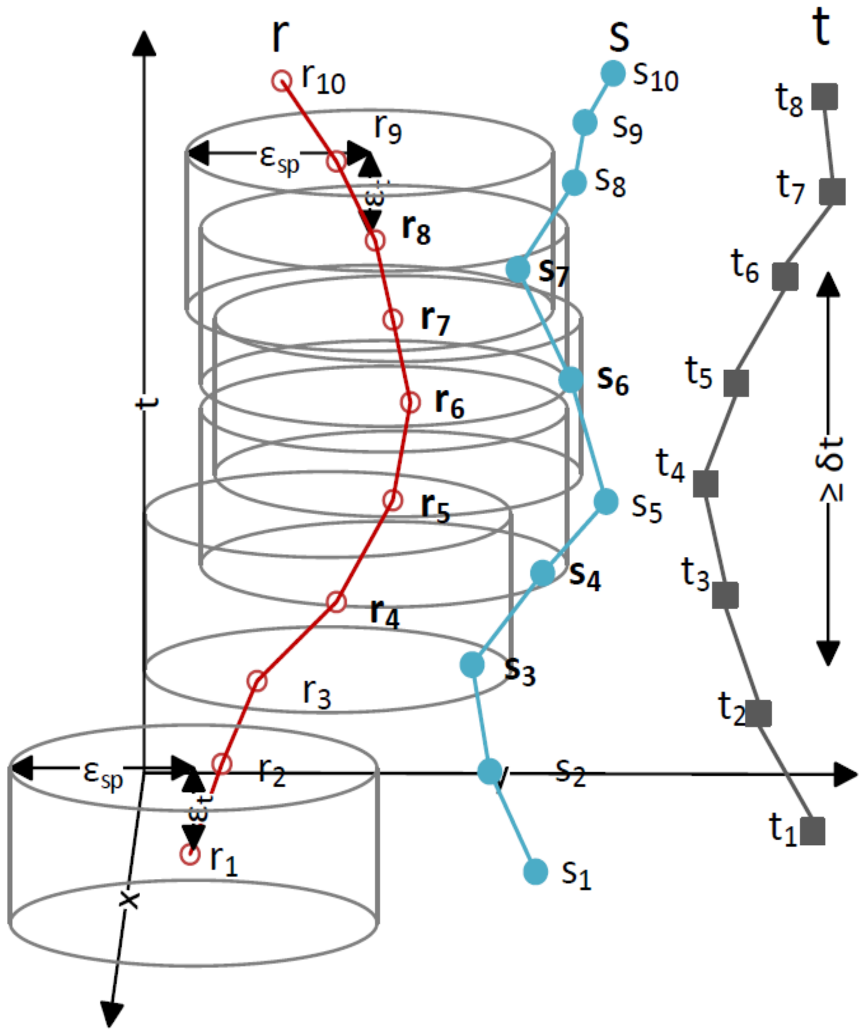
Definition *(Breaking points) A point $r_i \in r \in R$ is a breaking point iff it is not a joining point with any other point $s_j \in S$:*

$$\nexists s_j \in S\colon DistS(r_i, s_j) \leq \epsilon_{sp} \wedge DistT(r_i, s_j) \leq \epsilon_t.$$

Lemma *The set of breaking points is necessary in order to produce the correct result set for the Subtrajectory Join problem.*

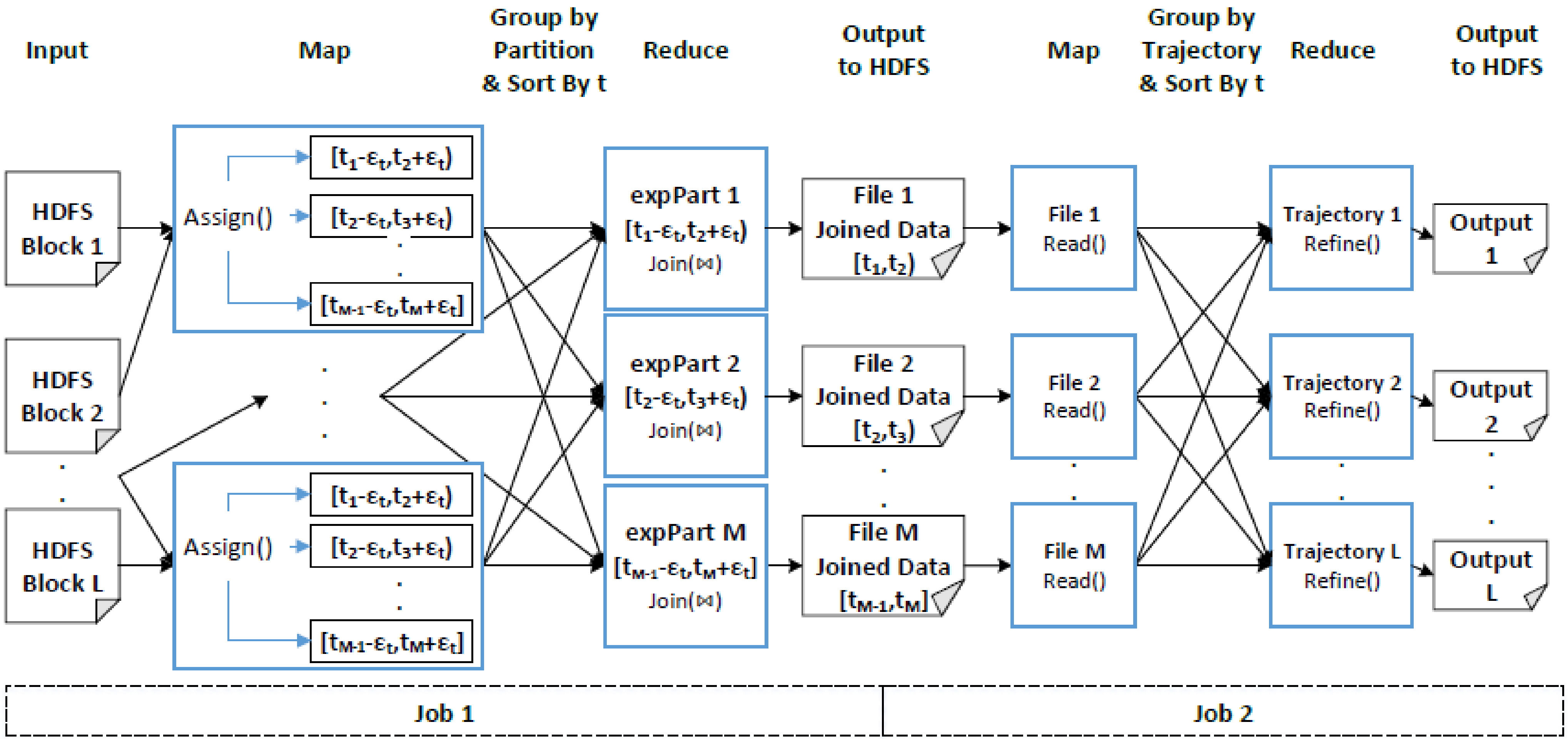
Definition *(Necessary subset sNJP of non-joining points) A pair of non-joining points (r_i, s_j), where $r_i \in r \in R$ and $s_j \in s \in S$, belongs to sNJP, iff (a) $\nexists$ a point $s_p \in s$, with $p \neq j$, such that s_p is a joining point w.r.t. r_i, (b) $\nexists$ a point $s_q \in s$, with $q \neq j$, such that $DistT(r_i, s_q) \leq DistT(r_i, s_j)$ and (c) at least one of the adjacent points of r_i, r_{i-1} or r_{i+1}, is a joining point w.r.t. some point $s_t \in s$, with $t \neq j$.*

Lemma *The set sNJP of pairs of non-joining points is necessary in order to produce the correct result set for the Subtrajectory Join problem.*



Distributed Subtrajectory Join on Massive Datasets

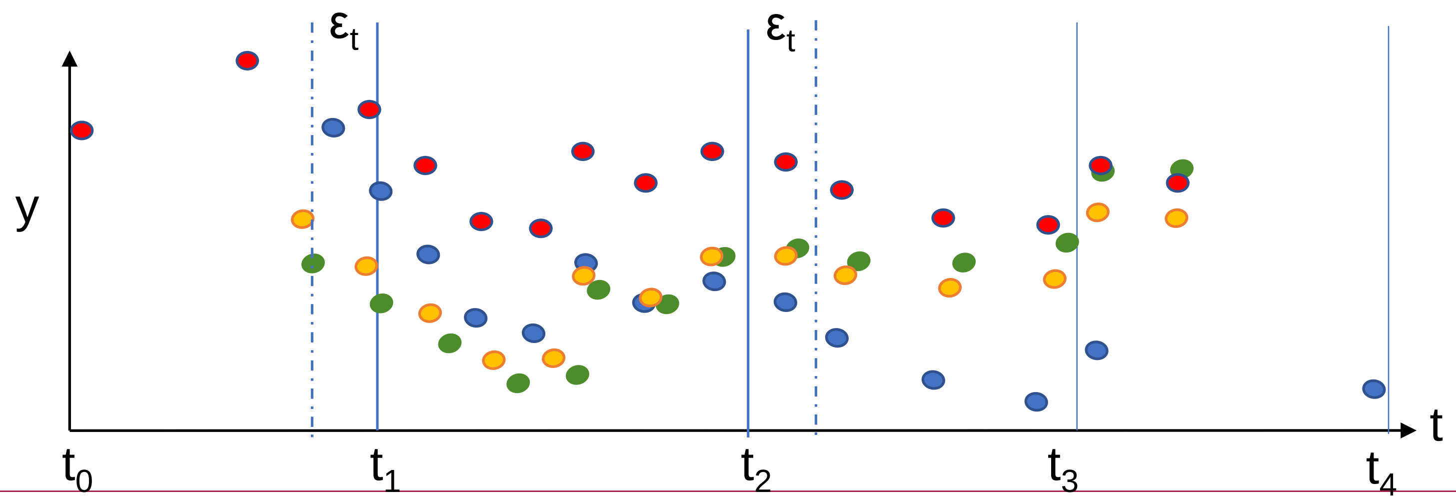
The Basic Subtrajectory Join Algorithm (DTJb)

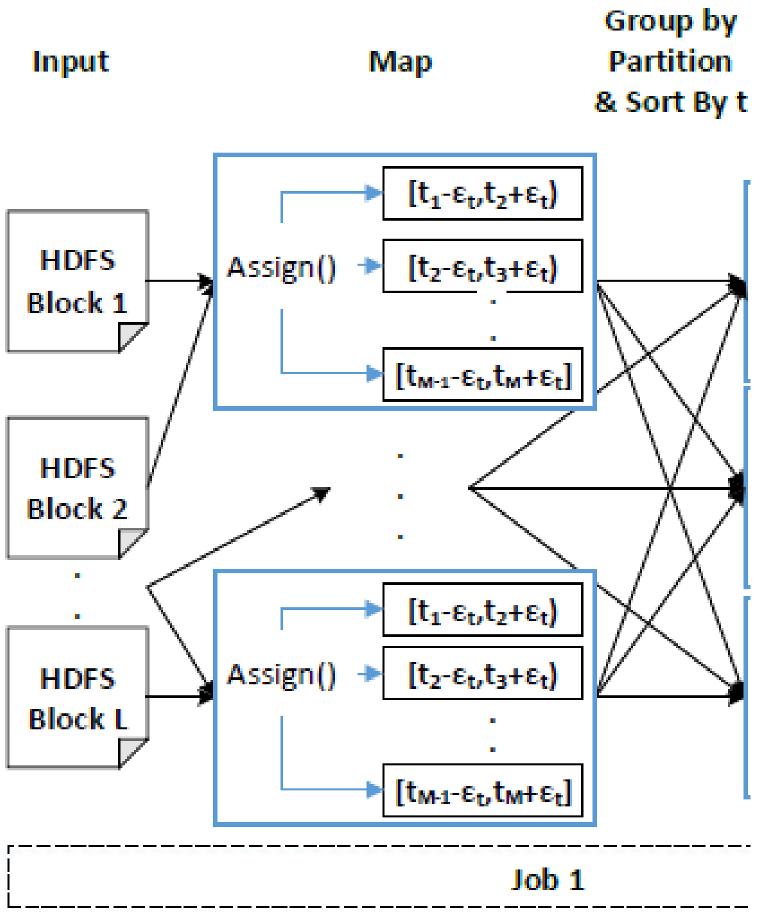


Distributed Subtrajectory Join on Massive Datasets

The Basic Subtrajectory Join Algorithm (DTJb) – Partitioning

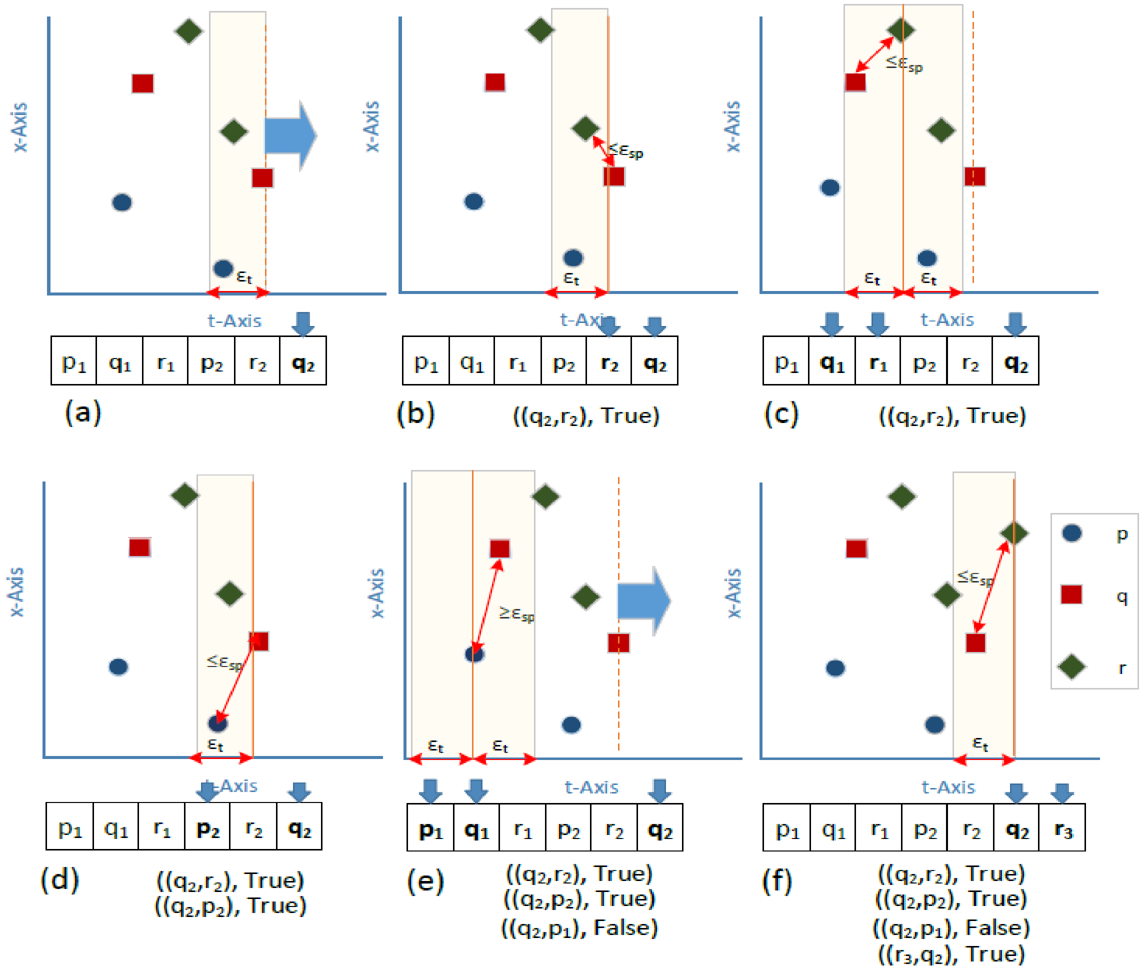
- *M* disjoint temporal partitions,
  - → cannot guarantee the correctness due to ε_t.
- We expand each partition by ε_t, so that it can be processed independently in parallel.
- Duplication avoidance





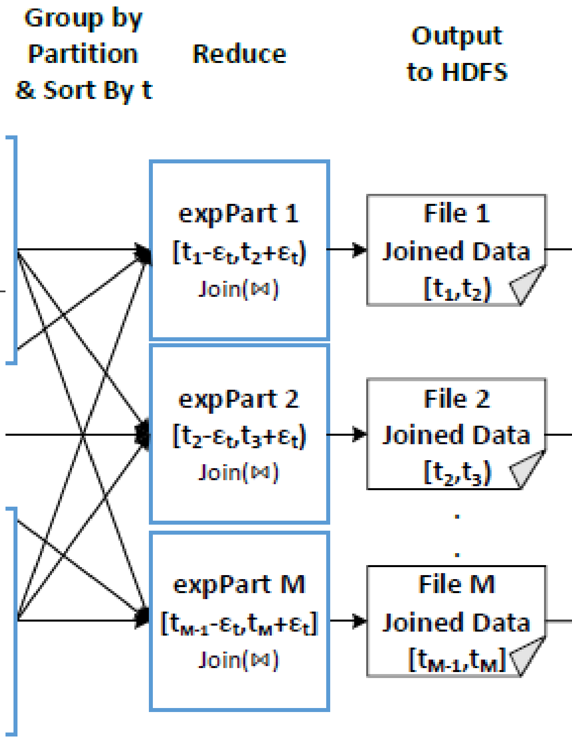
Distributed Subtrajectory Join on Massive Datasets

The Basic Subtrajectory Join Algorithm (DTJb) – Join



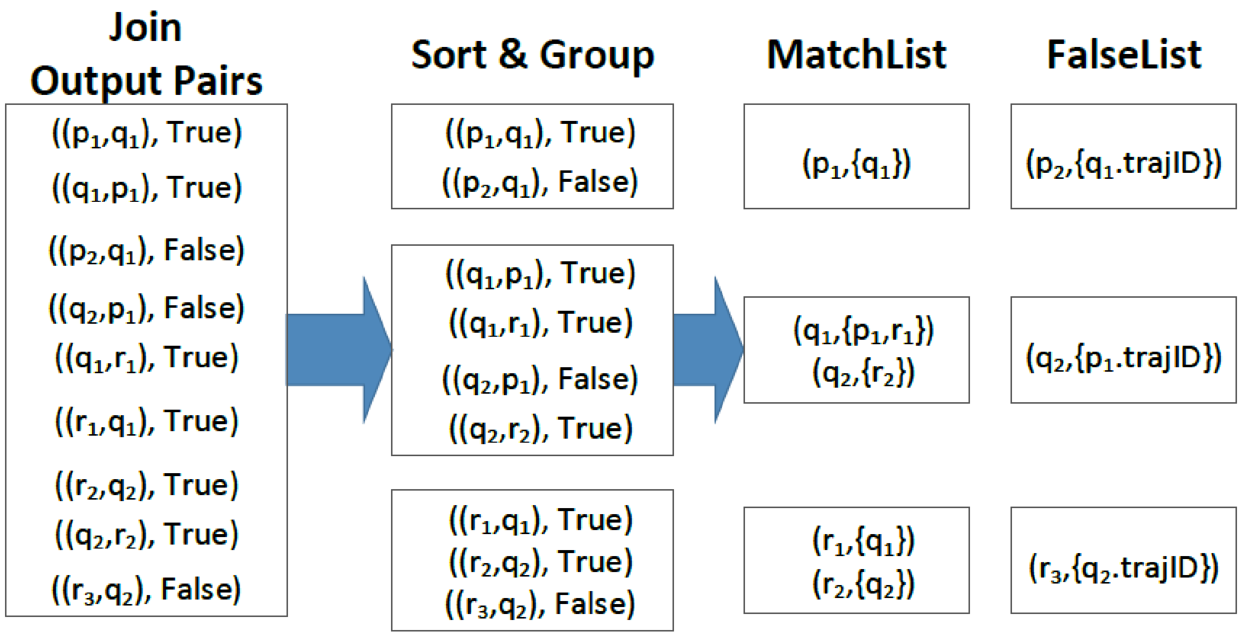
Algorithm Join($expPart, \epsilon_{sp}, \epsilon_t$)

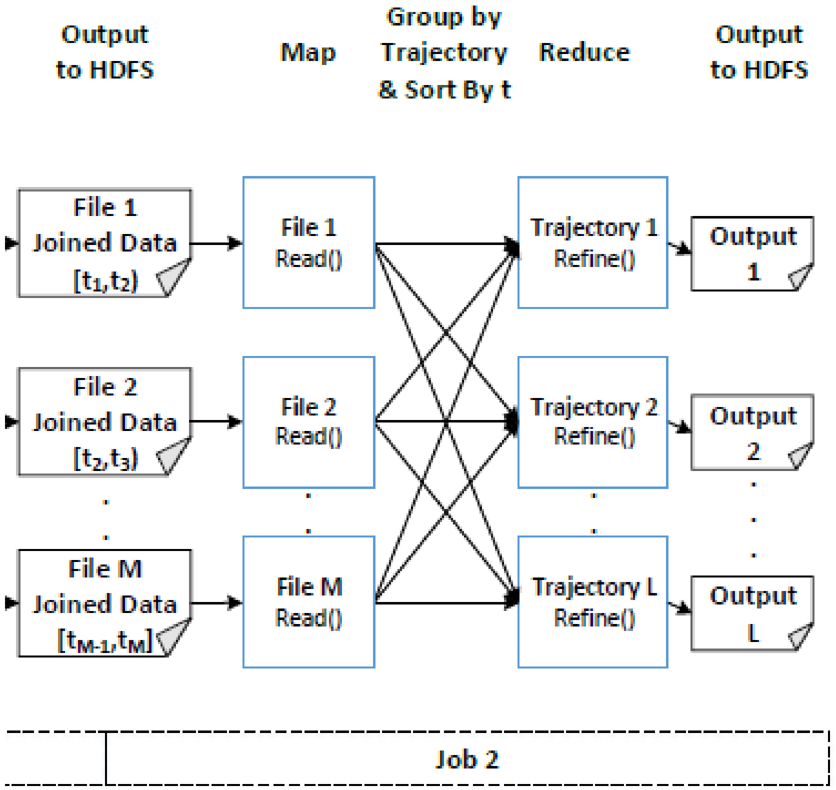
Input: An $expPart$, ϵ_{sp}, ϵ_t
Output: All pairs of JP, BP and candidate $sNJP$
for each $point\ i \in expPart$ **do**
  $D[i] \leftarrow point$
  TRJPlaneSweep($D[], \epsilon_{sp}, \epsilon_t$)
TreatLastTrPoints()
for each $point\ j \in BP[]$ **do**
  output(($BP[j], null$), True)



Distributed Subtrajectory Join on Massive Datasets

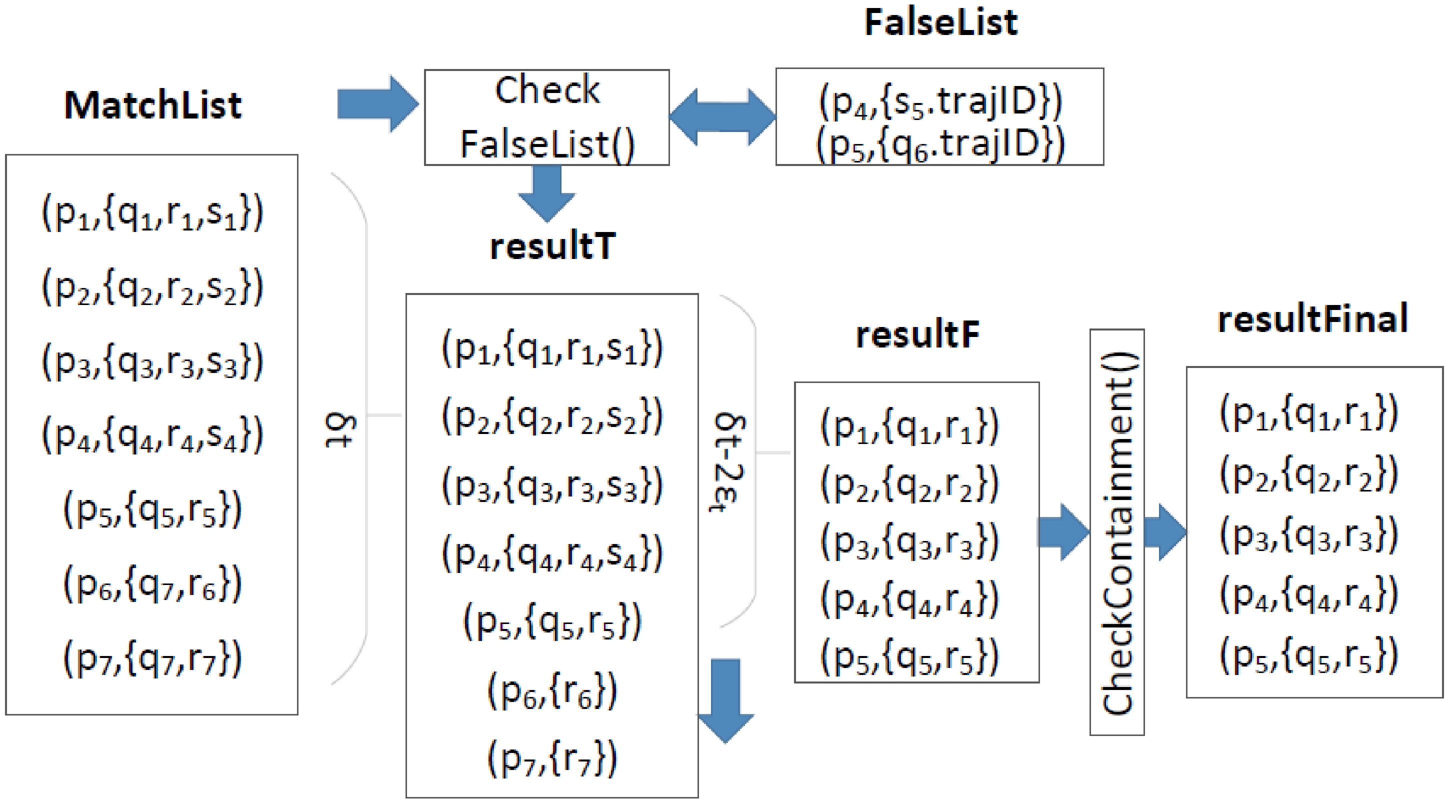
The Basic Subtrajectory Join Algorithm (DTJb) – Refine

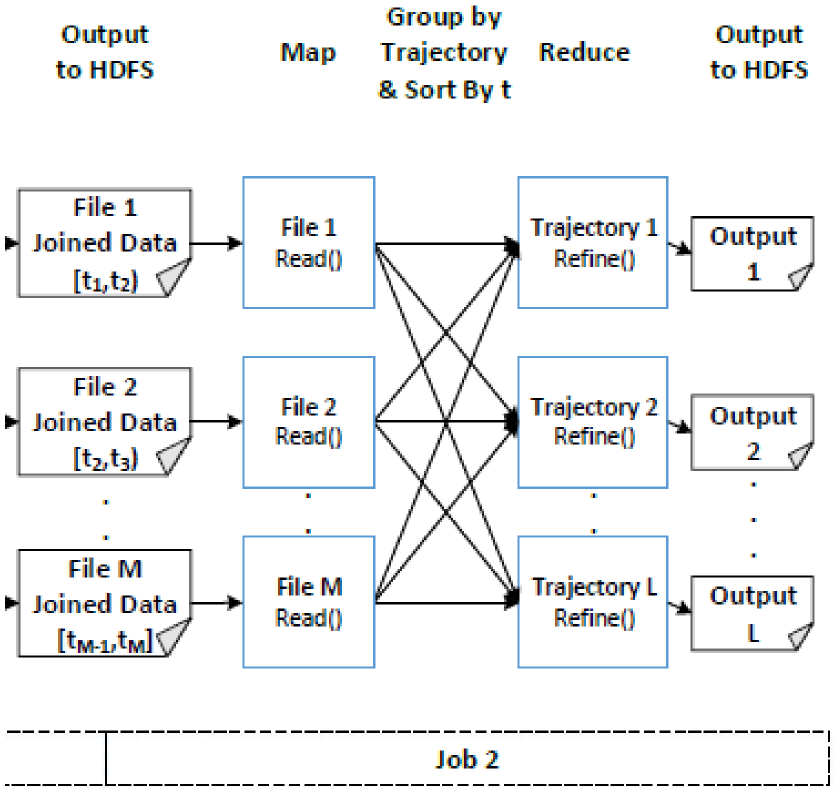




Distributed Subtrajectory Join on Massive Datasets

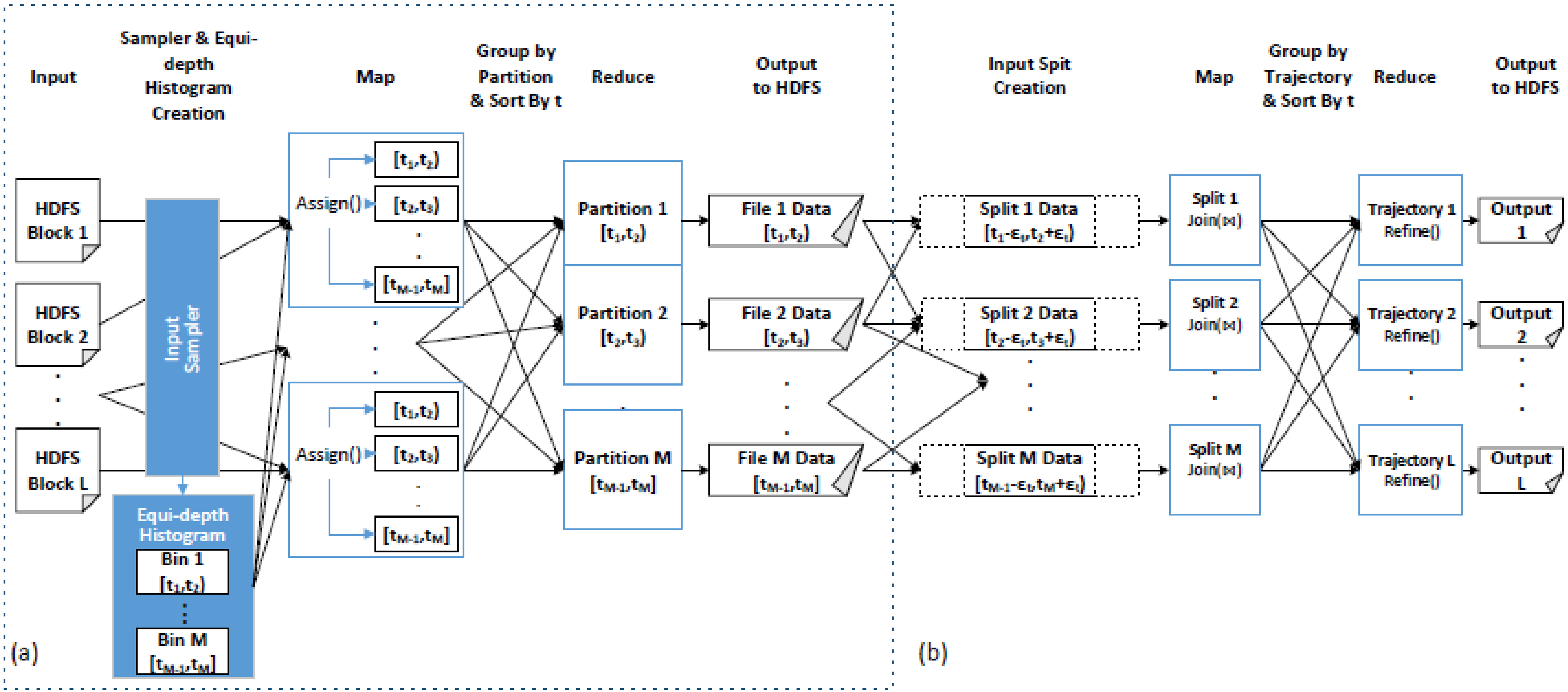
The Basic Subtrajectory Join Algorithm (DTJb) – Refine





Distributed Subtrajectory Join on Massive Datasets

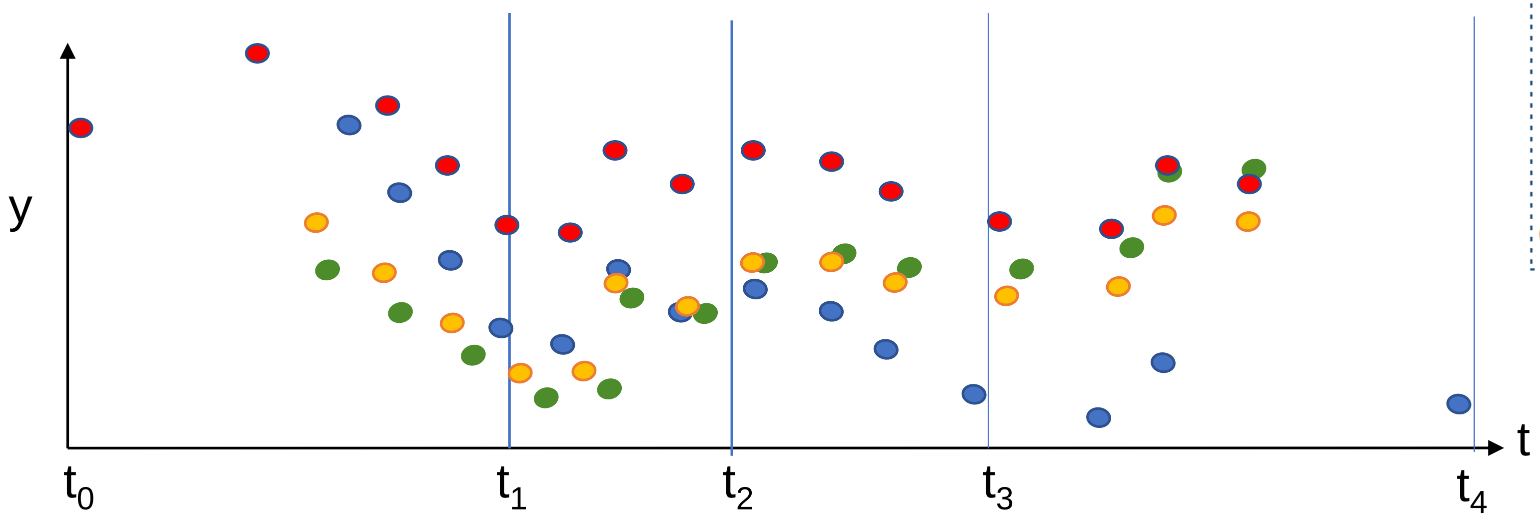
Subtrajectory Join with Repartitioning (DTJr)

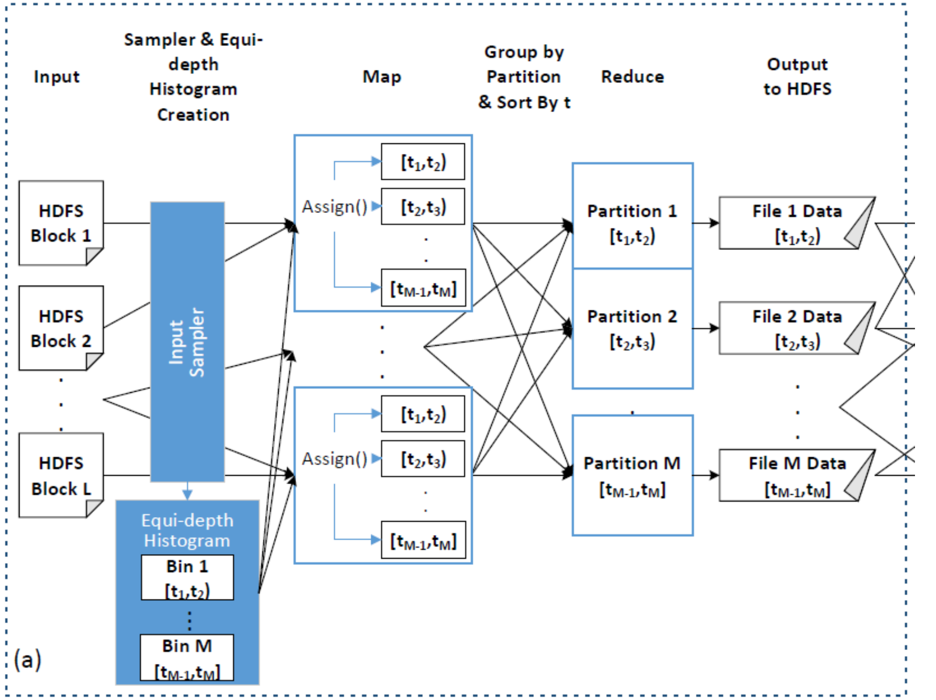


Distributed Subtrajectory Join on Massive Datasets

Subtrajectory Join with Repartitioning (DTJr) – Repartitioning

- We **sample** the input data (*InputSampler)* and construct an **equi-depth histogram** on the temporal dimension.
- The histogram contains *M* equi-sized bins.

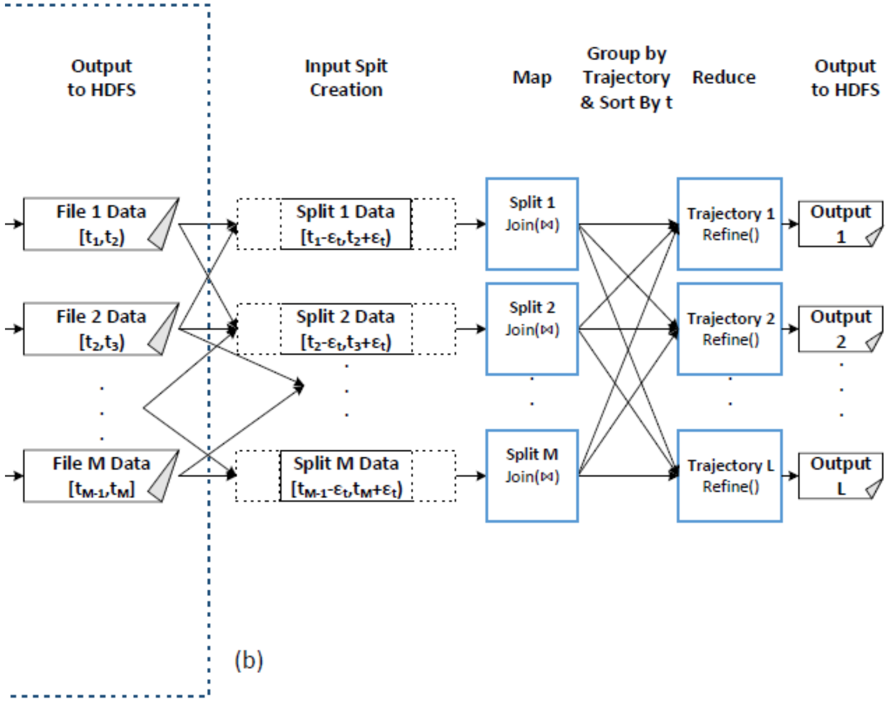


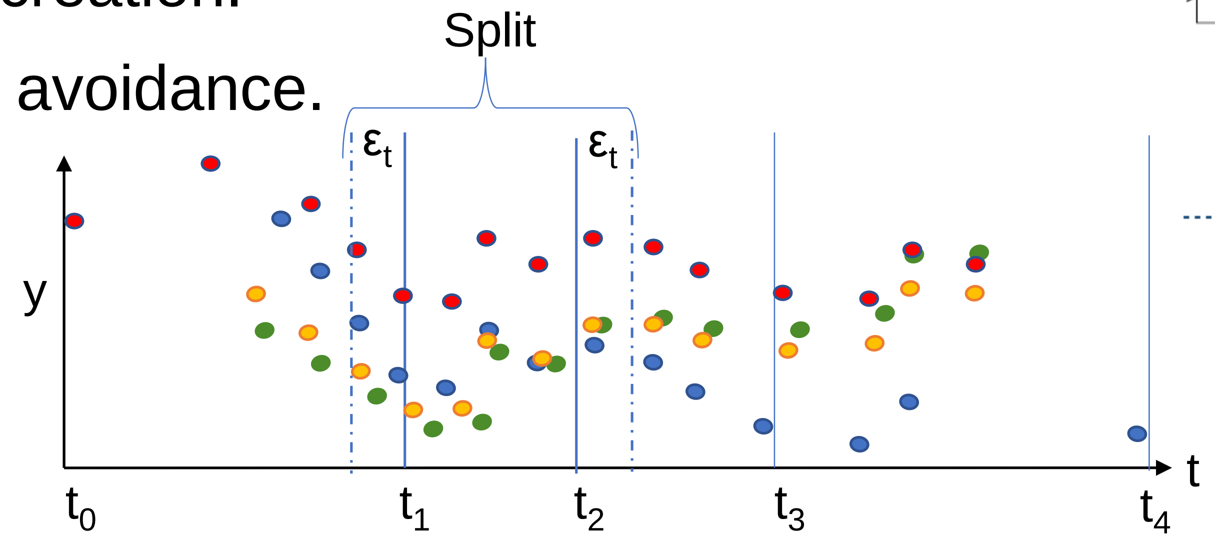


Distributed Subtrajectory Join on Massive Datasets

Subtrajectory Join with Repartitioning (DTJr) – Repartitioning

- In order to minimize the I/O cost,
  - *Join* procedure in the *Map* phase, and the
  - *Refine* in the *Reduce* phase.
- Each *HDFS* block is expanded with additional points that exist at time (+/-) ε_t, and this is the process of *InputSplits* creation.
- Duplication avoidance.

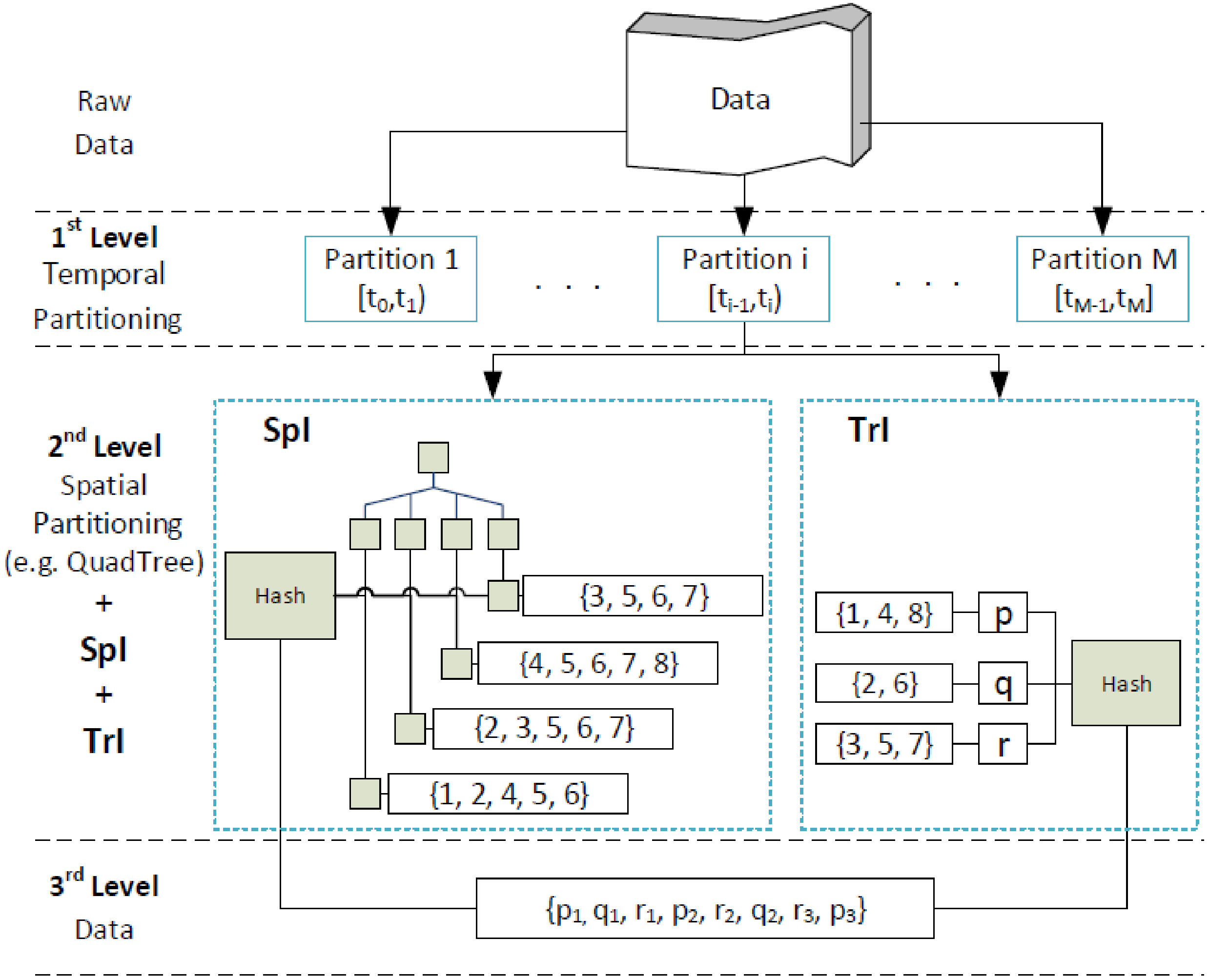




Distributed Subtrajectory Join on Massive Datasets

Index-based Subtrajectory Join with Repartitioning (DTJi)

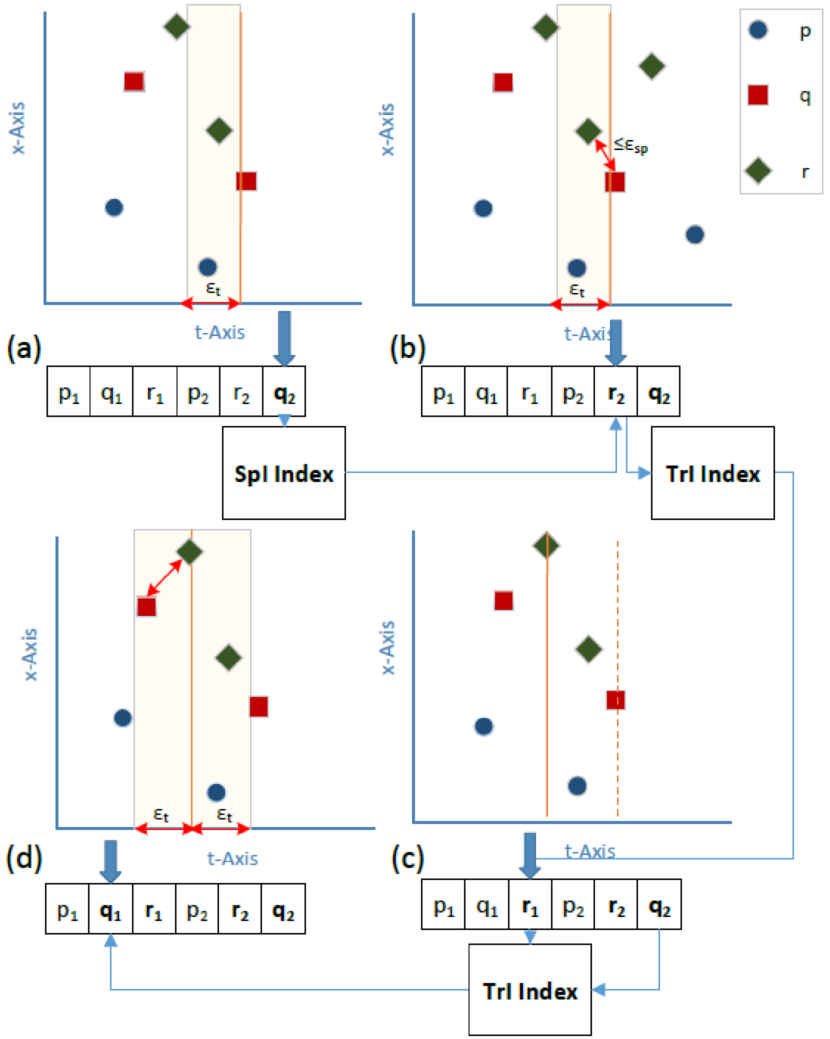
- Spatial Index (SpI)
  - QuadTrees
- Trajectory Index (TrI)
  - HashMap
    - Key → trajectory ID
    - Value → list of positions



Index-based Subtrajectory Join with Repartitioning (DTJi)

Algorithm 6.4 Join^I (Split, ϵ_{sp}, ϵ_t, t_s^{base}, t_e^{base})

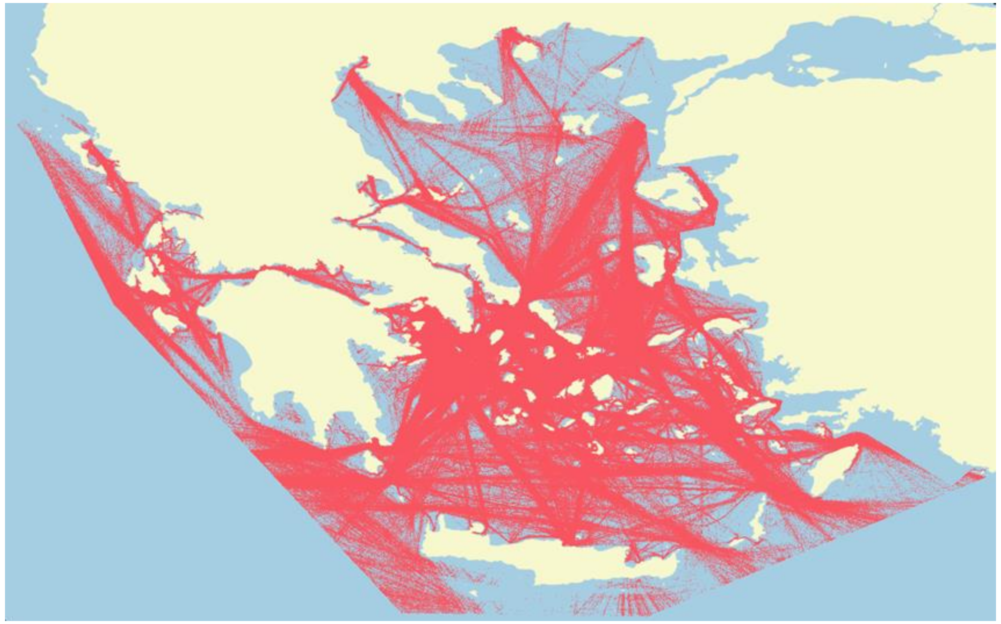
Input: A split, ϵ_{sp}, ϵ_t, t_s^{base}, t_e^{base}
Output: All pairs of JP, BP and candidate $sNJP$
$QT \leftarrow LoadQuadTree()$
for each $point\ i \in Split$ **do**
    if $point.t \in [t_s^{base} - \epsilon_t, t_e^{base} + \epsilon_t]$ **then**
        $D[i], TrI, SpI \leftarrow point$
    TRJPlaneSweep$^I(D[], TrI, SpI, \epsilon_{sp}, \epsilon_t, t_s^{base}, t_e^{base})$
TreatLastTrPoints()
for each $point\ j \in BP[]$ **do**
    output$((BP[j], null)$, True)



Distributed Subtrajectory Join on Massive Datasets

Experimental Study

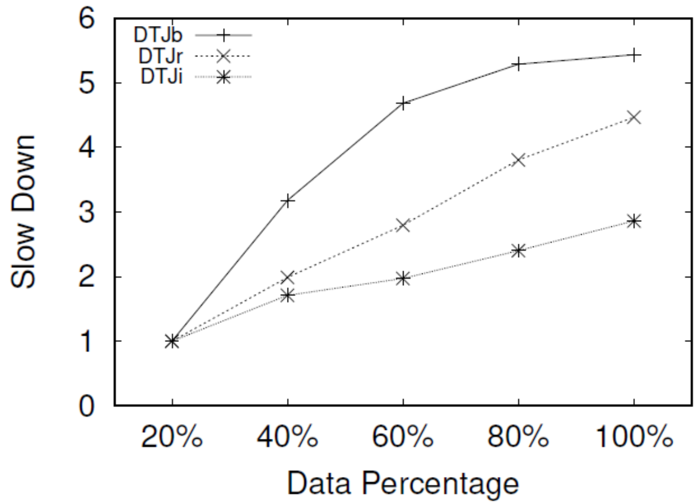
- Setting
  - 49 node Hadoop 2.7.2 cluster (1 Master + 48 Slaves).
  - Each slave → 4 CPU cores, 4 GB of RAM and 60 GB of HDD.
  - 192 containers.
- Dataset → IMIS
  - 699,031 trajectories of ships moving in the Eastern Mediterranean
  - for a period of 3 years.
  - This dataset contains approximately 1.5 billion records, 56GB in total size.

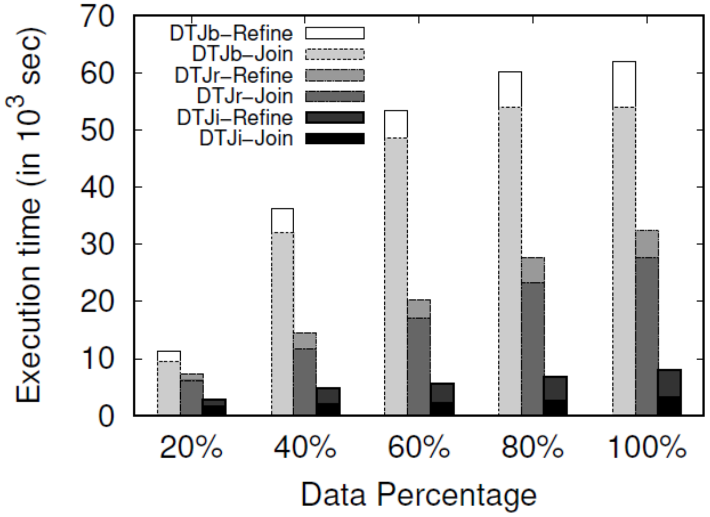


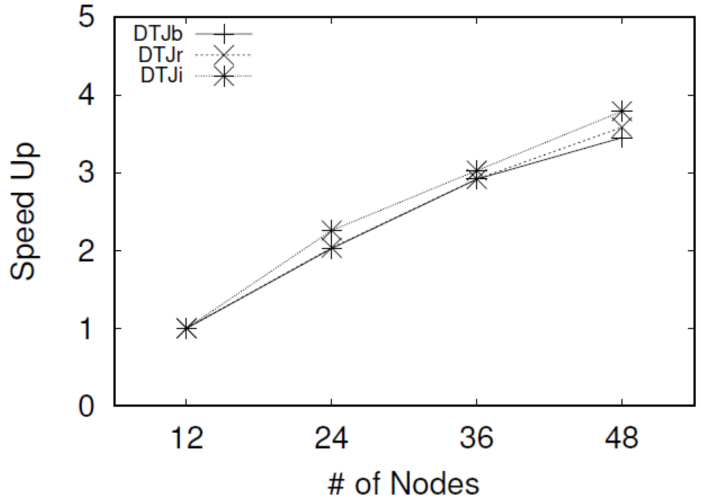
Statistic	# Trajectories	# Points	Area	Dataset Duration
IMIS	699031	1.5×10^9	Eastern Mediterranean	3 years

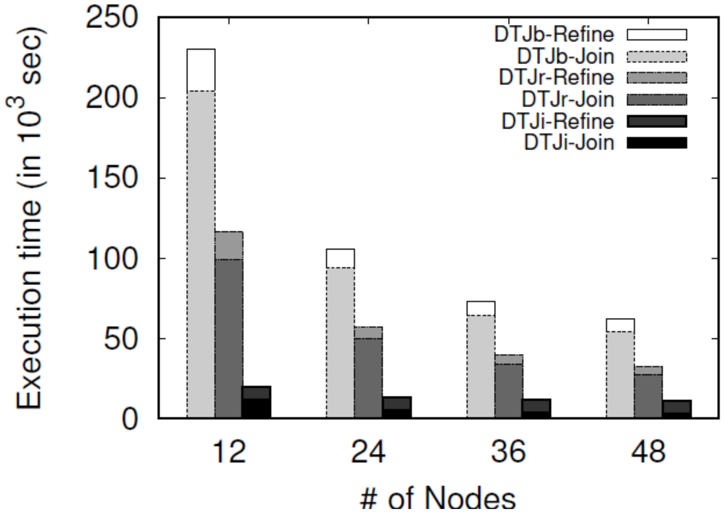
Distributed Subtrajectory Join on Massive Datasets

Experimental Study - Scalability



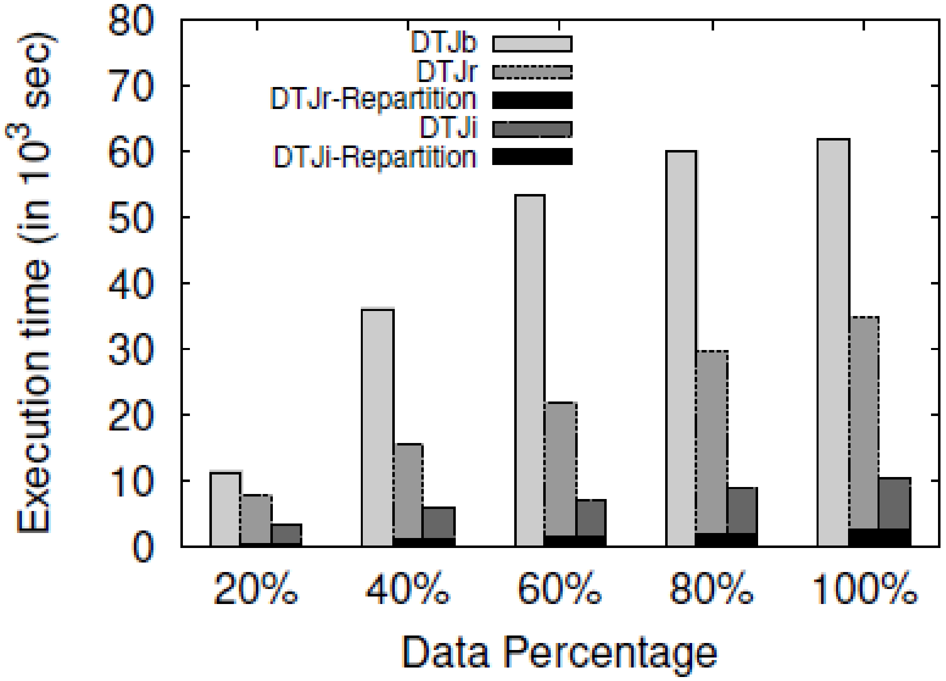


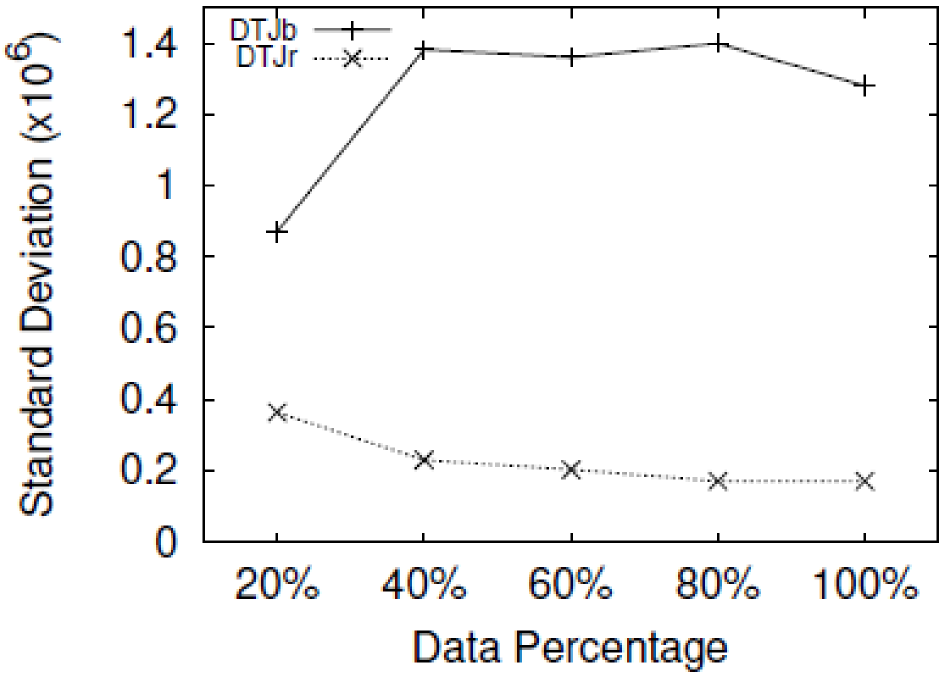




Distributed Subtrajectory Join on Massive Datasets

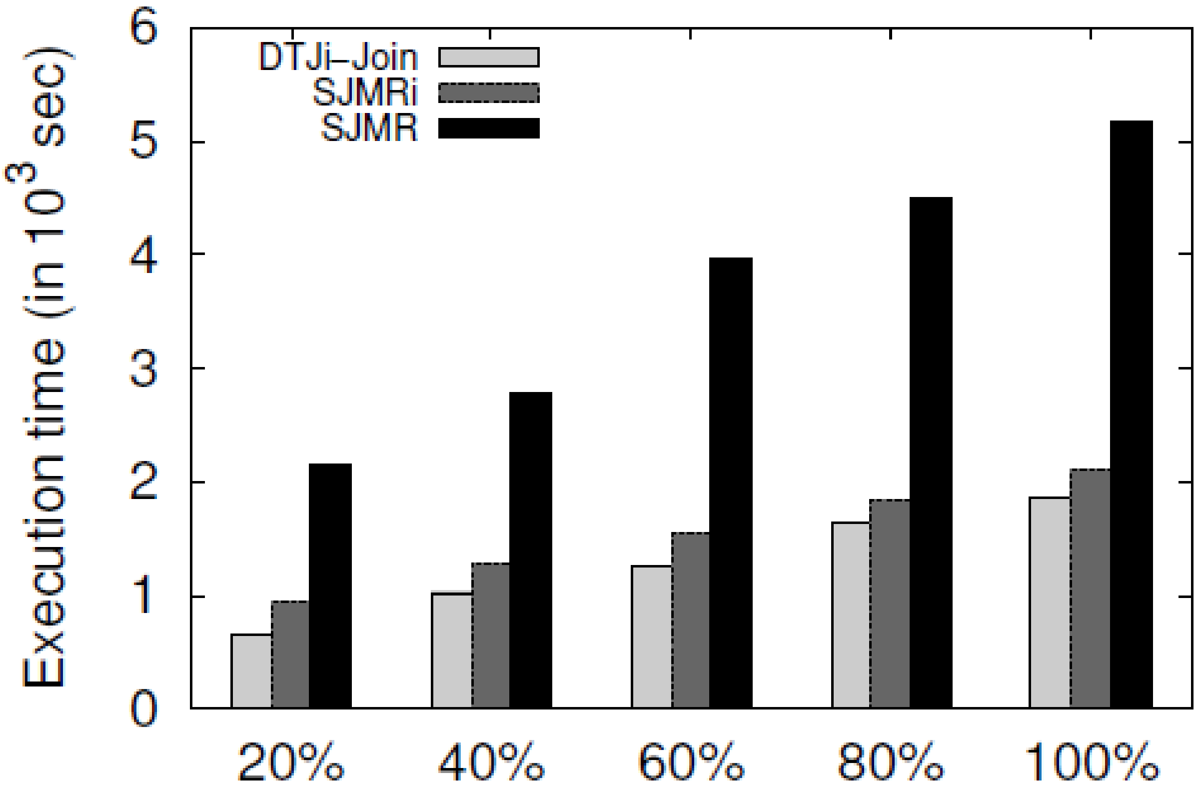
Experimental Study – Repartitioning and Load Balancing





Distributed Subtrajectory Join on Massive Datasets

Experimental Study – Comparative Evaluation



Distributed Subtrajectory Join on Massive Datasets

Summary

- We introduced the **Distributed Subtrajectory Join** query.
- We addressed it in a scalable manner following the MapReduce programming model.
- The results show that our index-based solution performs up to 16x faster compared with our baseline and 3x faster than the closest related state of the art algorithm.

Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

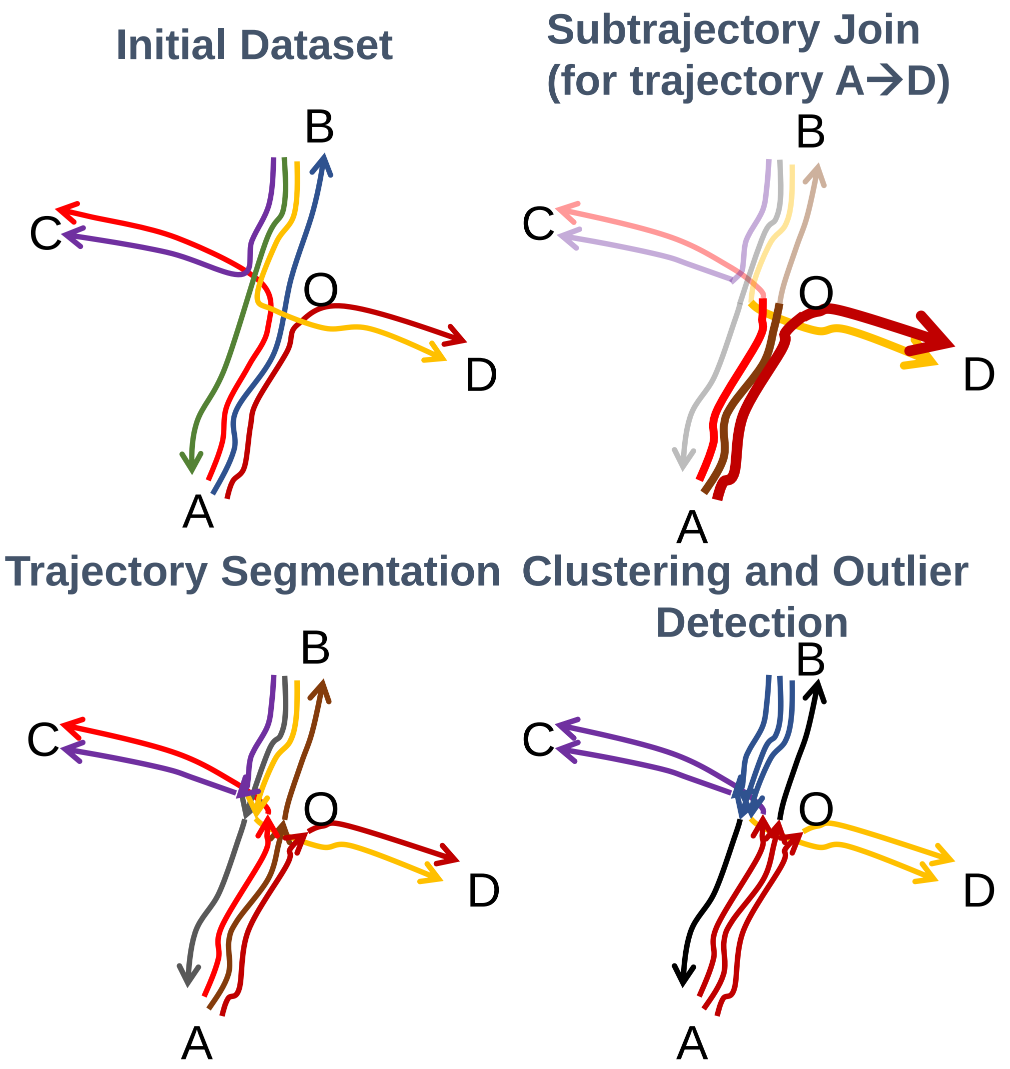
Scalable Distributed Subtrajectory Clustering

Problem Formulation

- The problem of Subtrajectory Clustering can be decomposed to
  1. Subtrajectory Join → Computation of LCSS
  2. Trajectory Segmentation
  3. Subtrajectory Clustering
- Assuming a cluster is represented by its representative (or medoid) subtrajectory, we define clustering as an optimization problem:

$$SSCR = \sum_{\forall R_i \in R} \sum_{\forall r'_j \in C_i} Sim(R_i, r'_j)$$

- Distributed Subtrajectory Clustering → Solve Problems 1, 2 and 3 (in this order) in a parallel/distributed way.

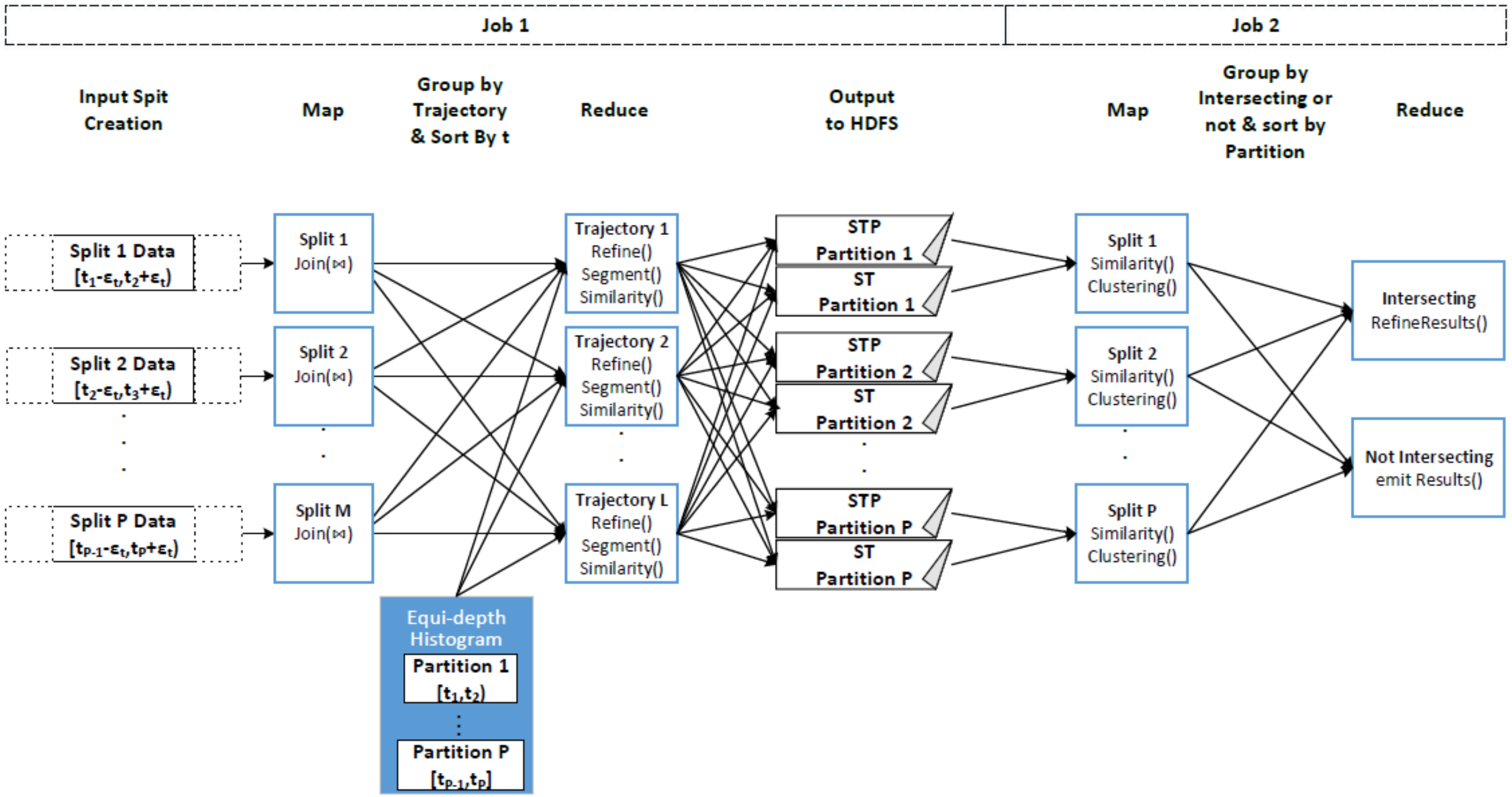


Scalable Distributed Subtrajectory Clustering

Problem Solution – Overview

```
Algorithm    DSC(D)
Input: D
Output: set C of clusters, set O of outliers
Preprocessing: Repartition D;
for each partition D_i ∈ ∪_{i=1}^{P} D_i do
    perform Point-level Join;
group by Trajectory;
for each Trajectory r ∈ D do
    perform Subtrajectory Join;
    perform Trajectory Segmentation;
group by D_i;
for each subtrajectory r' ∈ D_i do
    calculate Similarity with other subtrajectories;
perform Clustering;
perform Refine Results;
return C and O;
```

$\mathbf{DTJ_i}$ (lines 3–8)

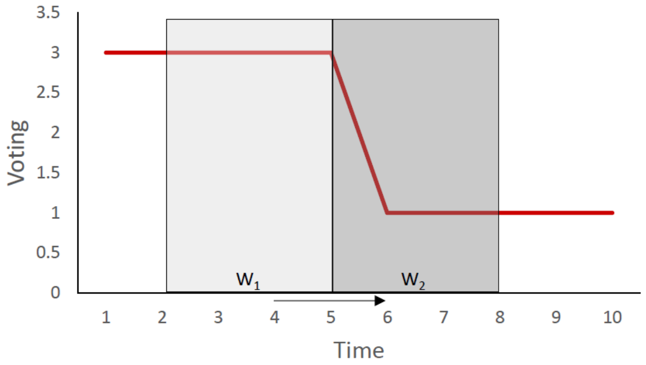


Scalable Distributed Subtrajectory Clustering

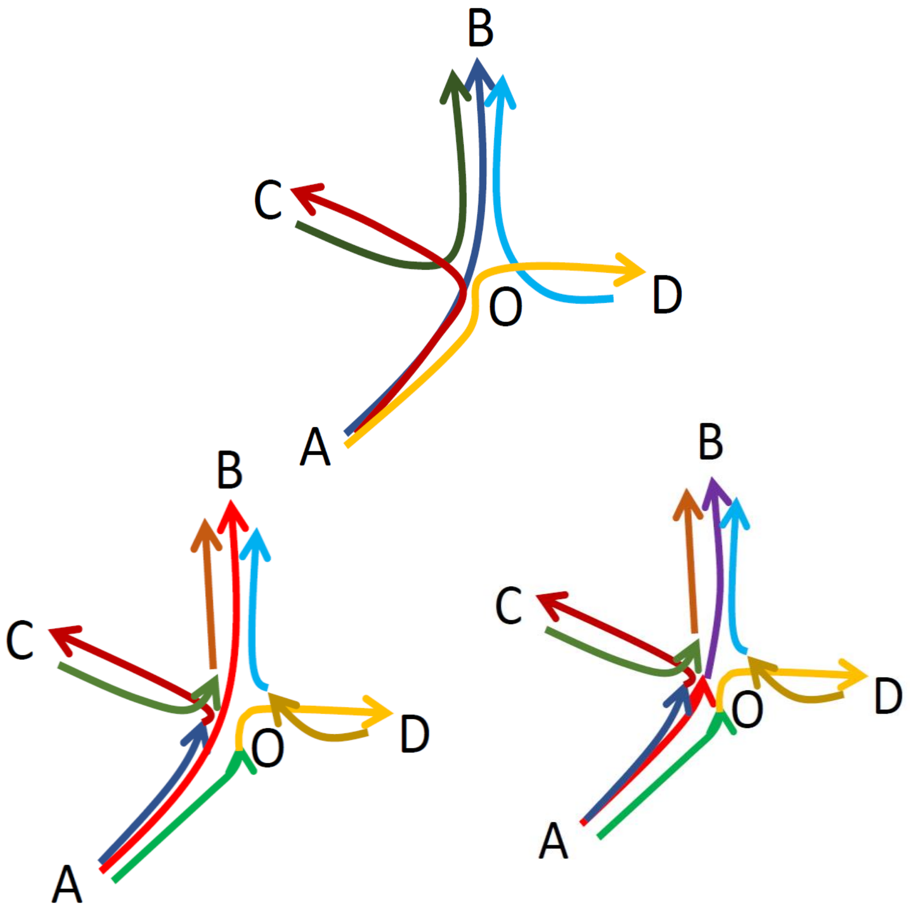
Problem Solution – Distributed Trajectory Segmentation

- $TSA_1 \rightarrow$ identifies the beginning of a new subtrajectory whenever the density ($V(r_i)$) of its neighborhood changes significantly.

$$V(r_i) = \sum_{\forall s \in D} \frac{d_s(r_i, s_k)}{\epsilon_{sp}}$$



- $TSA_2 \rightarrow$ identifies the beginning of a new subtrajectory whenever the composition of its neighborhood changes substantially.

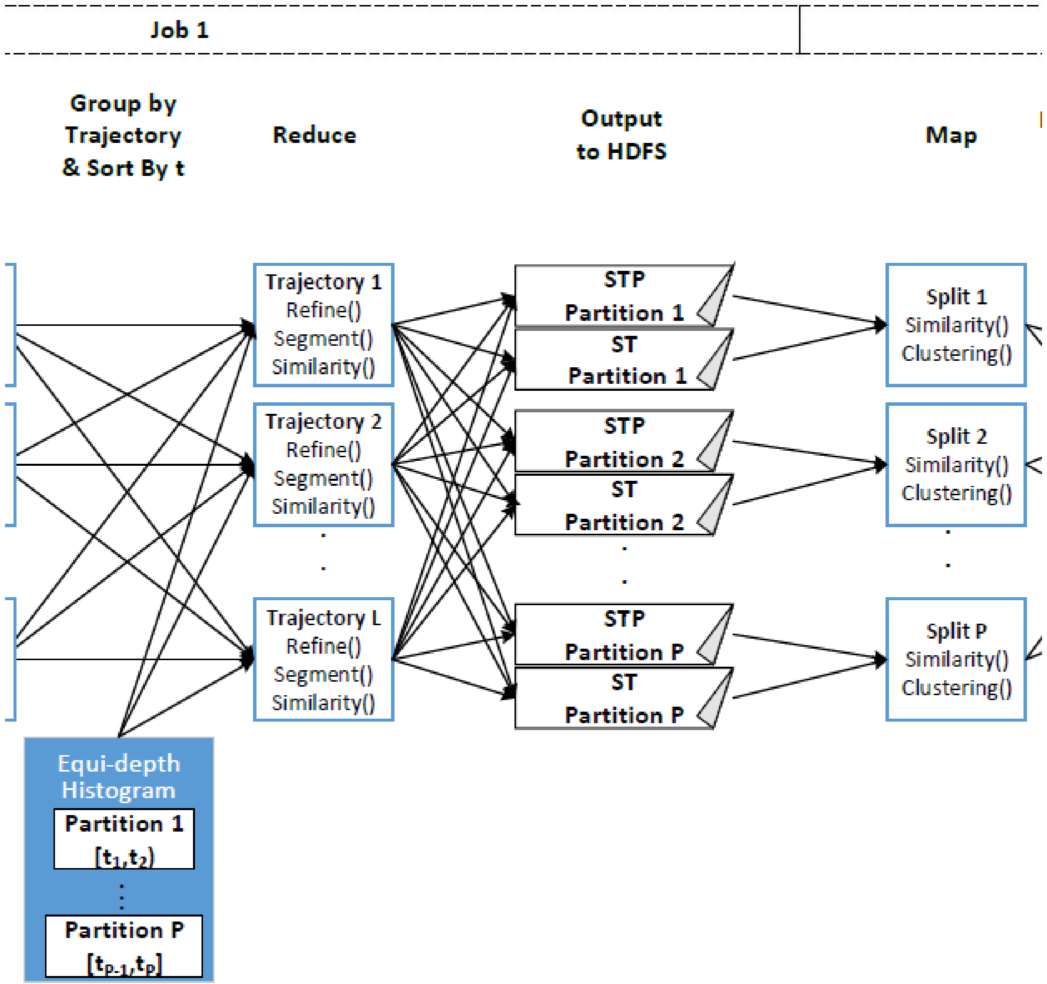


Scalable Distributed Subtrajectory Clustering

Problem Solution – Distributed Clustering

Algorithm $Clustering(SP, ST, k, \alpha)$

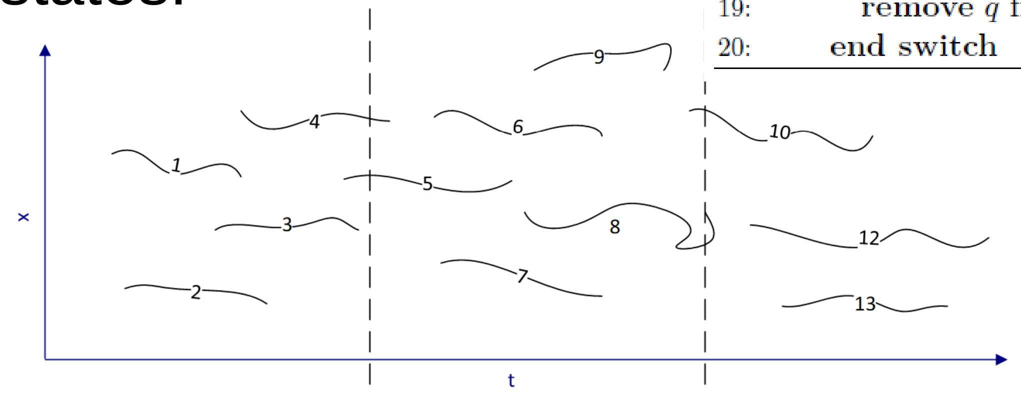
Input: SP, ST, k, α
Output: set C of clusters, set O of outliers
sort ST by V in descending order;
for each element $st \in ST$ **do**
  if $st \notin R$ **then**
    if $st.V \geq k$ **then**
      $st \to R$;
      for each element $l \in st.AdjLst$ **do**
        if $l \notin C$ **then**
          if $Sim(l, st) \geq \alpha$ **then**
            $l \to C(st)$;
            if $l \in O$ **then**
              $O = O - l$;
          else
            $O = O \cup l$
        else
          if $Sim(l, st) > Sim(l, R(l))$ **then**
            $C(R(l)) = C(R(l)) - l$;
            $l \to C(st)$;
    else
      $O = O \cup st$;
$C = C \cup R$



Scalable Distributed Subtrajectory Clustering

Problem Solution – Refinement of Results

- For each partition, the clustering procedure will decide whether a subtrajectory is
  - a *Representative* (*R*),
  - a *Cluster Member* (*C*) or
  - an *Outlier* (*O*).
- For each intersecting subtrajectory *q* and for each pair of consecutive partitions (*i, j*) with which *q* intersects, *q* can have the following pairs of states:
  - (a) *O-O*,
  - (b) *R-R*,
  - (c) *C-C*,
  - (d) R-C (C-R),
  - (e) R-O (O-R) and
  - (f) C-O (O-C)



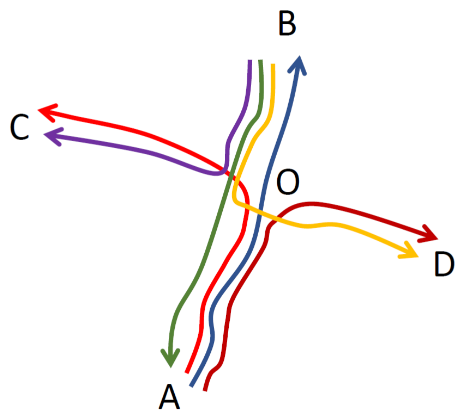
Algorithm $RefineResults(q)$

Input: *Intersecting Subtrajectories*
Output: set C of clusters, set O of outliers
for each pair p $\rightarrow (P_i, P_{i+1})$ of Partitions **do**
  $P_i \cap P_{i+1} \rightarrow I$
  for each element $e \in I$ **do**
    switch (p)
    case (a)**:**
      remove q **from** O_i;
    case (b)**:**
      merge $C_i(q)$ **and** $C_{i+1}(q)$;
    case (c)**:**
      if $Sim(q, R_i(q)) > Sim(q, R_{i+1}(q))$ **then**
        remove q **from** C_{i+1};
      else
        remove q **from** C_i;
    case (d)**:**
      remove q **from** C;
    case (e),(f)**:**
      remove q **from** O;
    end switch

Scalable Distributed Subtrajectory Clustering

Experimental Study – Datasets

- Intersection
- Brest
- SIS



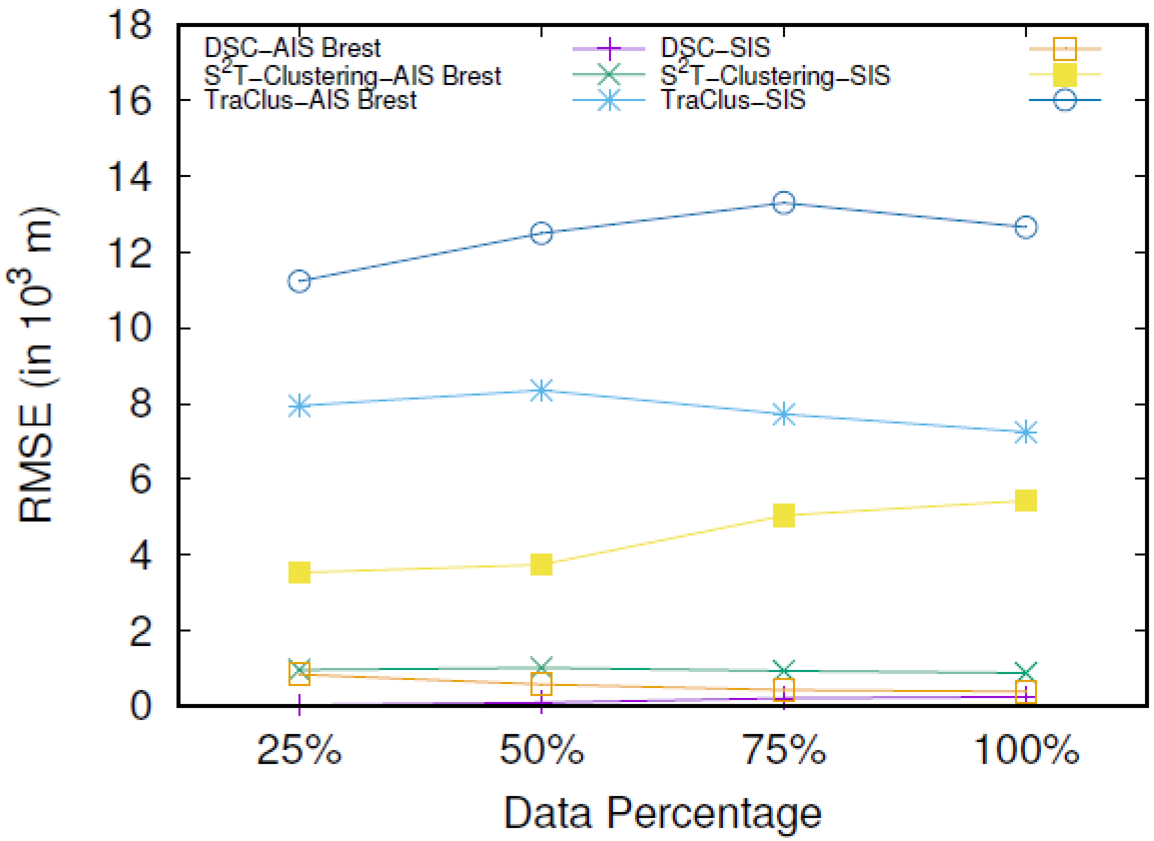
Statistic	Intersection
# Trajectories	409
# Points	2573
Dataset Duration	23 seconds

Statistic	# Trajectories	# Points	Area	Dataset Duration
Brest	365000	17×10^6	Brest	6 months
SIS	2.2×10^7	7.2×10^8	Rome and Tuscany	2.5 years

Scalable Distributed Subtrajectory Clustering

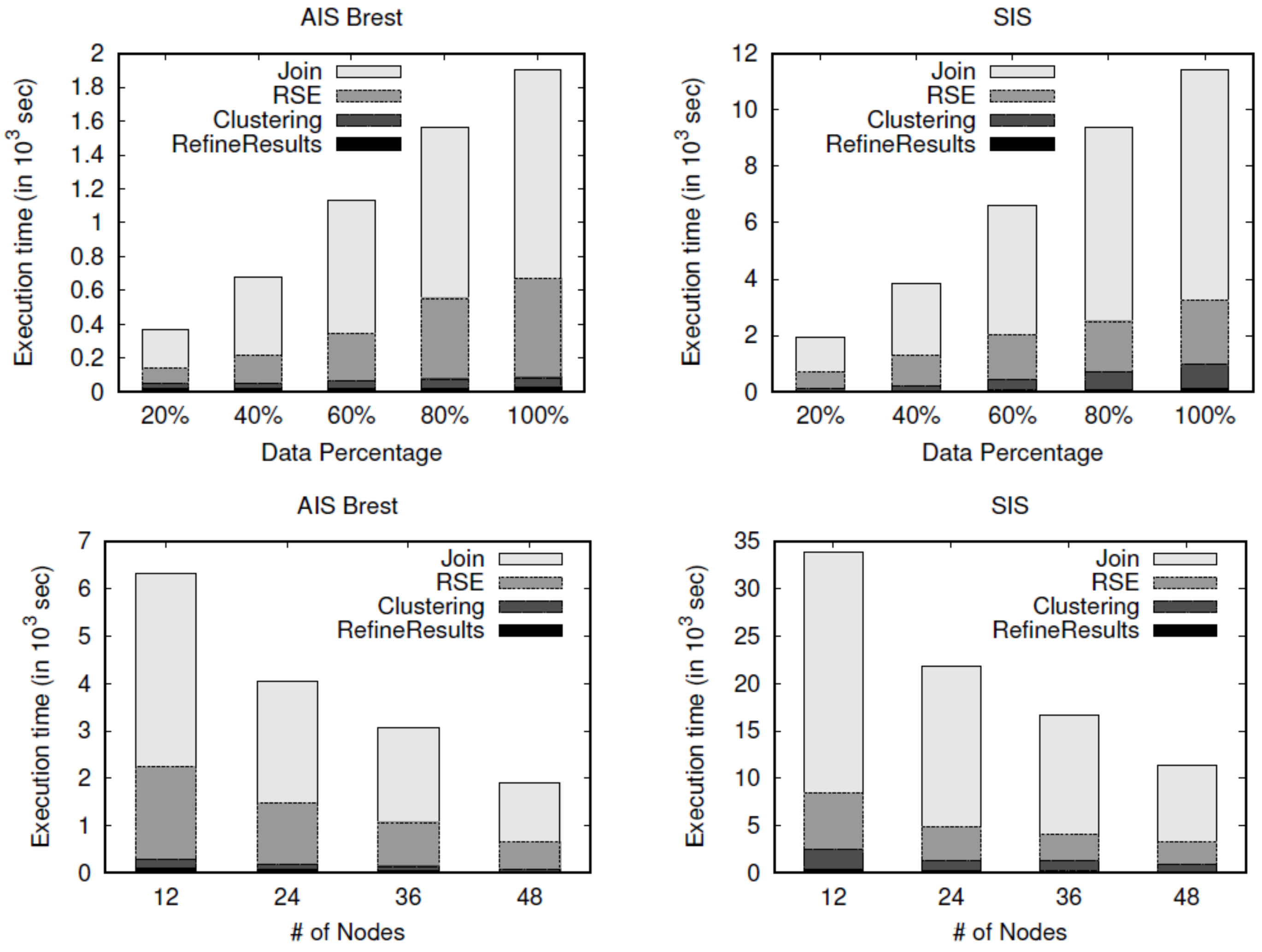
Experimental Study – Comparison with Related Work

- DSC vs S^{2}T-Clustering vs TraCluS.



Scalable Distributed Subtrajectory Clustering

Experimental Study – Performance and Scalability



Scalable Distributed Subtrajectory Clustering

Summary

- We addressed the problem of **Distributed Subtrajectory Clustering** by building upon a scalable subtrajectory join query operator.
- We proposed two alternative trajectory segmentation algorithms.
- We proposed a distributed clustering algorithm where the clusters are identified in a parallel manner and get refined as a final step.
- Our experimental study shows that our solution is more effective and far more scalable (since it is distributed) from the state of the art.

Publications related to Part II

Originating from this PhD Thesis

1. P. Tampakis, C. Doulkeridis, N. Pelekis, and Y. Theodoridis. **Distributed Subtrajectory Join on Massive Datasets**. ACM Trans. Spatial Algorithms and Systems, to appear.
2. P. Tampakis, N. Pelekis, C. Doulkeridis, and Y. Theodoridis. **Scalable Distributed Subtrajectory Clustering**. In Proceedings of IEEE Big Data Conf., 2019.

Other publications “influenced” by the above

1. P. Petrou, P. Nikitopoulos, P. Tampakis, A. Glenis, N. Koutroumanis, G. M. Santipantakis, K. Patroumpas, A. Vlachou, H. Georgiou, E. Chondrodima, C. Doulkeridis, N. Pelekis, G. L. Andrienko, F. Patterson, G. Fuchs, Y. Theodoridis, G. A. Vouros. **ARGO: A Big Data Framework for Online Trajectory Prediction**. In Proceedings of SSTD, 2019.
2. P. Petrou, P. Tampakis, H. Georgiou, N. Pelekis, Y. Theodoridis. **Online Long-term Trajectory Prediction based on Mined Route Patterns**. In Proceedings of ECML/PKDD Workshops, 2019.

Part IV
Outlook

Outline

- Setting the Scene
  - Motivation & Application Scenarios
  - Challenges & Contributions
  - Datasets
- In-DBMS Centralized Algorithms and Techniques
  - In-DBMS Sampling-based Subtrajectory Clustering
  - Temporal-constrained Subtrajectory Cluster Analysis
  - Time-Aware Subtrajectory Clustering in Hermes@PostgreSQL
- Distributed Algorithms and Techniques
  - Distributed Subtrajectory Join on Massive Datasets
  - Scalable Distributed Subtrajectory Clustering
- Outlook
  - Conclusions & Ideas for Future Work

Conclusions

The Thesis' contributions, in a nutshell:

- **Centralized environment**:
  - S^2T-Clustering, an efficient subtrajectory clustering algorithm
  - *ReTraTree*, an indexing scheme assisting *temporal-constrained subtrajectory cluster analysis*
  - *QuT-Clustering*, an in-DBMS query operator over *ReTraTree*
- **Distributed (Big Data) environment**:
  - algorithms addressing the *Distributed Subtrajectory Join (DTJ)* query, following the MapReduce programming model
  - algorithms addressing the *Distributed Subtrajectory Clustering problem,* by building upon *DTJ*

Ideas for Future Work

- Concerning the problem of **temporally-constrained subtrajectory cluster analysis**:
  - investigate **real-time solutions**, on big data architectures.
- Regarding the **Distributed Subtrajectory Join** operator:
  - investigate how the solution provided can be applicable to **streaming trajectories**
  - examine how this query can be extended and utilized in order to be able to **identify efficiently various mobility patterns** (e.g., flocks, convoys, moving clusters swarms etc.)
  - investigate the potential of extending the solution proposed here to tackle the problem of **k-nn trajectory join**.
- Concerning the problem of **Distributed Subtrajectory Clustering**:
  - extend our solution with properties of **density-based clustering** algorithms
  - investigate the possibility of addressing the same problem in a **streaming** environment, since our algorithm employs a **single pass** over the data.

References

1. M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Indexing spatiotemporal archives. *VLDB J.*, 15(2):143–164, 2006.
2. H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1):1068–1080, 2008.
3. P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, pages 364–381, 2005.
4. Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1):723–734, 2010.
5. M. Nanni and D. Pedreschi. Time-focused clustering of trajectories of moving objects. *J. Intell. Inf. Syst.*, 27(3):267–289, 2006.
6. J. Lee, J. Han, and K. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.
7. P. K. Agarwal, K. Fox, K. Munagala, A. Nath, J. Pan, and E. Taylor. Subtrajectory clustering: Models and algorithms. In *PODS*, pages 75–87, 2018.
8. P. Bakalov and V. J. Tsotras. Continuous spatiotemporal trajectory joins. In *GSN*, pages 109–128, 2006.
9. Y. Chen and J. M. Patel. Design and evaluation of trajectory join algorithms. In *SIGSPATIAL*, pages 266–275, 2009.
10. H. Ding, G. Trajcevski, and P. Scheuermann. Efficient similarity join of large sets of moving object trajectories. In *TIME*, pages 79–87, 2008.
11. N. Ta, G. Li, Y. Xie, C. Li, S. Hao, and J. Feng. Signature-based trajectory similarity join. *IEEE Trans. Knowl. Data Eng.*, 29(4):870-883, 2017.
12. N. Pelekis, P. Tampakis, M. Vodas, C. Panagiotakis, Y. Theodoridis. In-DBMS Sampling-based Sub-trajectory Clustering, In Proceedings of EDBT, 2017.
13. N. Pelekis, P. Tampakis, M. Vodas, C. Doulkeridis Y. Theodoridis. On Temporal-Constrained Sub-Trajectory Cluster Analysis, Data Mining and Knowledge Discovery, 31(5):1294-1330, 2017.
14. Hermes@PostgreSQL MOD engine. URL: http://infolab.cs.unipi.gr/hermes.
15. G. L. Andrienko, N. V. Andrienko, P. Bak, D. A. Keim, and S. Wrobel. *Visual Analytics of Movement*. Springer, 2013.
16. M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In ICDE, pages 673–684, 2002.

Questions ?