Sampling-based Sub-Trajectory Clustering

Information Management Lab (InfoLab)
Department of Informatics
University of Piraeus

Nikos Pelekis
Costas Panagiotakis
Marios Vodas
Panagiotis Tampakis
Yannis Theodoridis

August 2013
# Table of Contents

1. INTRODUCTION .................................................................................................................. 3
2. RELATED WORK .................................................................................................................... 5
3. OVERVIEW OF $S^2T$-CLUSTERING .................................................................................. 6
   3.1 Voting and segmentation into sub-trajectories ............................................................... 7
   3.2 Sub-trajectory sampling, clustering and outlier detection ............................................... 8
4. THE PROPOSED FRAMEWORK ............................................................................................. 9
   4.1 A Constraint-aware Sub-trajectory Clustering Algorithm ............................................ 11
   4.2 A note on complexity analysis ....................................................................................... 13
   4.3 $S^2T$-Clustering as an efficient query operator in a MOD ........................................... 13
5. EXPERIMENTAL RESULTS ............................................................................................... 16
   5.1 Verification of the Ground Truth .................................................................................. 16
REFERENCES ........................................................................................................................... 19
1. INTRODUCTION
Knowledge discovery over mobility data [13] exposes patterns of moving objects that can be exploited in several fields. For example, in both mature (transportation, climatology, zoology, etc.) and emerging domains, such as mobile social networks, scientists work with mobility (mostly GPS) data, resulting in Moving Object Databases (MOD) consisting of trajectories of moving objects. Although during the last decade, there have been significant achievements in the field [13], researchers continuously call for new methods aiming at deeper comprehension and analysis of mobility data.

In the literature of mobility data mining one can identify several types of mining models used to describe various collective behavioral patterns. As such, there are works that propose classification schemes for trajectory data [28], identify various types of clusters of moving objects [12][31][27][38], moving clusters [30][22], sequential trajectory patterns [14], flocks [5], convoys [21], swarms [29], outliers [26] and others.

Focusing on trajectory clustering, the majority of related work proposes different distance functions, which are utilized by well-known clustering algorithms to identify collective behavior among whole trajectories [31][38][36]. More interestingly, a counterpart line of research tries to discover local patterns in MOD, i.e. patterns that exist only for a part of the lifespan of the moving objects. Some of these techniques simplify the given trajectories, focusing on spatial criteria and ignoring the temporal dimension [28][27]. Other proposals either fail to capture the complex nature of the trajectories that follow arbitrary motion patterns [27], or focus on the discovery of very specific definitions of movement patterns [5][15][16][21][29].

A relevant line of research aims to build representatives out of a trajectory dataset, by either generating artificial data [27][38] or by sampling the dataset itself [39][35]. Inspired by the above, in this report we study an important problem for MOD analysis and mining, that of sub-trajectory clustering and outlier detection, which will be modeled as an optimization problem. More specifically, in our approach we seek for a methodology that segments and selects those sub-trajectories from a MOD that preserve as much as possible the properties and the mobility patterns hidden in the original MOD. This sampled subset of sub-trajectories is chosen with the objective to cover as much as possible the space-time extent of the MOD, and will serve as the seeds around which the clusters will be identified. Note that this approach is the reverse of the traditional clustering methodology that first groups data in clusters and then identifies the proper representatives of each cluster [27].

Fig. 1 illustrates an example of the clustering of our methodology applied to a MOD comprised by four trajectories, $T_1, ..., T_4$. (In this figure, the time dimension has been ignored for visualization reasons.) Our proposal will be able to identify two clusters of sub-trajectories (in red and blue) and five outlier sub-trajectories (in black). On the other hand, the current state-of-the-art technique for sub-trajectory clustering, (TRACLUS [27]) will first simplify the trajectories into large line segments and then apply a grouping of similar line segments, thus, as delineated by the authors, discovering linear patterns only and failing to identify complex (e.g. snake-like) patterns like the ones that appear in Fig. 1. In other words, when applied to this toy MOD, TRACLUS would eventually result in the discovery of six linear clusters (one cluster for each time the snake-like motion changes direction). On the contrary, according to our approach, the clustering uses the results of segmentation and sampling processes that take into account the spatiotemporal similarity. In addition, there is no constraint on the complexity of the shape of sub-trajectories found, yielding a clustering that is related only on line segment representativeness. Furthermore, we should note that we do not pose any geometrical constraints in terms of parameters of the proposed methodology as those required by moving clusters [30][22], flocks [5][16], convoys [21], swarms [29]. This type of parameters, such as the radius of disc, or the (max) duration and the cardinality of the patterns are hard to be defined, while their tuning may result in completely diverging outcomes.
Figure 1: (a) a MOD of four trajectories; (b) two identified clusters of sub-trajectories (in red and blue) and five outlier sub-trajectories (in black).

The design of such a clustering methodology that will be free of shortcomings like the above, is subject to two more requirements which challenged our research: (a) we seek for an efficient and scalable solution, which (b) should be based not on an ad hoc implementation using a sophisticated access method, but rather on a real-world MOD management system so as to be useful in real-world application scenarios, where concurrency and recovery issues should be taken into consideration. Both requirements ask for a MOD engine, such as SECONDO [1] and HERMES [37]. Although enhancing such engines with mining operators is challenging and has not been addressed in the literature, their use is subject to an important drawback, which is a shortcoming of the indexing extensibility interface of the corresponding ORDBMSs on which they are implemented; see [24] for a discussion on this issue. Indexing shortcomings have been addressed by the introduction of the Generalized Search Tree (GiST) [19], which, however, has not been utilized in the context of mobility data management and, more specifically, in MOD engines. In the current work, for the first time to the best of our knowledge, the GiST interface is used to index mobility data in order to fulfill the initially posed requirements.

In the current work, we exploit on the voting and segmentation-sampling algorithms proposed in [35] upon which we devise our clustering methodology. More specifically, we employ a voting process that allows us to describe the “representativeness” of each segment of a trajectory in a MOD as a smooth continuous descriptor, and an algorithm for the automatic segmentation of trajectories into “homogenous” sub-trajectories according to the values of “representativeness”. Subsequently, we employ a deterministic sampling procedure that selects only those sub-trajectories that optimally describe the entire MOD. At the final step of our approach, we devise an automatic method for sub-trajectory clustering driven by the aforementioned “representative” sample of sub-trajectories. A nice feature of our approach is that a user may either trust on the algorithm to detect the optimal number of clusters or request a specific number of clusters.

Although optimality and deterministic-ness (as prescribed in the last two afore-mentioned steps of our approach) are desirable properties, sometimes may ignore real-world requirements. To exemplify such a requirement, consider the case where an analyst would like to pose certain constraints to be fulfilled by the members of a cluster (e.g. their cardinality should be at least $K$, i.e. to satisfy $K$-anonymity), even this is contradicting to optimality. In order to support such cases we also propose a constraint-aware variant of our methodology, which handles the sub-trajectory clustering problem from a constraint-based clustering perspective.

The contributions and merits of our work are summarized below:

- the problem of sub-trajectory clustering in a MOD is formulated as an optimization problem;
- a novel methodology (and the resulting algorithm, called $S^2T$-Clustering) is proposed to tackle the problem of sub-trajectory clustering, driven by a deterministic sampling methodology, with the number of clusters being automatically provided by the algorithm;
a constraint-aware variant of S²T-Clustering, which behaves as a template algorithm allowing any user-defined constraints to be taken into account during the clustering process; and

• an efficient implementation of S²T-Clustering as a query operator in a MOD engine, based on access methods designed using the indexing extensibility interface of GiST in PostgreSQL, in order to speed up the clustering process.

The rest of the report is organized as follows: Section 2 presents related work, Section 3 formulates the problem of sub-trajectory clustering and proposes our methodology, Section 4 provides an efficient, step-by-step realization of the methodology together with an analysis of its time complexity and a proposal on an efficient access method that speeds up the overall process. Experimental results that evaluate the proposed framework are provided in Section 5.

2. RELATED WORK
In this section, we review related work on mobility data mining (focusing on clustering) and related access methods for mobility data.

Recently, several approaches try to make well-known mining algorithms operational to trajectories. The common building block of these approaches is the use of different similarity functions as the mean to group trajectories into clusters. An interesting approach, which is also adopted by our work, is proposed in [10] for the efficient processing of most-similar trajectory (MST) queries. A similar distance function is used in T-OPTICS algorithm [31] (and its variant TF-OPTICS, which focuses on the discovery of the temporal intervals that lead to best clustering results), where the OPTICS [4] clustering algorithm, is made applicable for trajectory data. The previously mentioned temporal intervals are given by the user, which is a limitation of the approach, so TF-OPTICS reapplies T-OPTICS on portions of trajectories, that all live in exactly the same temporal period. The best out of the possible clusterings is chosen by applying some qualitative measures. Our approach can be viewed as a generalization of this approach, as we automatically identify patterns of sub-trajectories in an un-supervised way, which may have various non-predefined lifespans.

In [7] the authors proposed probabilistic techniques based on EM algorithm for clustering short trajectories using regression mixture models. This approach also aims on performing global clustering of whole trajectories and the notions of segmentation, sampling and sub-trajectory pattern mining is out of scope. In [38] the authors proposed a variant of FCM algorithm for MOD, called CenTR-I-FCM. The approach makes use of local patterns in time dimension as the base to identify global clusters of whole approximate-symbolic trajectories. Again the discovered local patterns are predefined with respect to their lifespan.

In [27] the authors proposed a partition-and-group framework for clustering 2-D trajectories which enables the grouping of similar sub-trajectories, based on a trajectory partitioning algorithm that uses the minimum description length principle. In its core it uses a variant of the DBSCAN algorithm that operates on the partitioned directed line segments. This work was the first to tackle the problem of identifying sub-patterns in mobility data, and, although similar in principle with our approach, it presents certain limitations as discussed earlier on the example of Fig. 1.

An interesting line of research include works that aim to discover several types of collective behavior among moving objects like flocks, leadership, convergence, encounter and sub-trajectory clusters patterns [5][16][25][15], moving clusters [22] convoys [21], and swarms [29]. Although these approaches provide lucid definitions of the mined patterns, their main limitation is that they are rather rigorous and sensitive to parameters, while their computation raises efficiency issues.

Our approach also finds commonalities to well-known approaches of clustering algorithms of point (vector) data [45][41], which first sample the dataset and then start the clustering process (aiming at high efficiency). Of course, it is not only that these vector-based algorithms are not applicable to MOD (due to the complex structure and properties of mobility data), there is also an essential difference between those techniques and our approach: while those rely on random sampling, in our approach the clustering is driven by a sample resulted by an optimization formula, thus leading to a deterministic solution of the sub-trajectory clustering problem.

The previous algorithms usually handle the issue of efficiency by employing a general-purpose access method (e.g. an R-tree-like structure), which however is implemented ad hoc and outside a DBMS or a
3. OVERVIEW OF S^2T-CLUSTERING

Let \( D = \{ T_1, T_2, \ldots, T_N \} \) be a MOD consisting of \( N \) trajectories. We assume that the objects are moving in the \( xy \)-plane. Let \( p_{ki} = (x_{ki}, y_{ki}, t_{ki}) \) be the \( i \)-th point, \( i \in \{1, 2, \ldots, L_k\} \), of the \( k \)-th trajectory \( T_k \), \( k \in \{1, 2, \ldots, N\} \), where \( L_k \) denotes the number of points \( T_k \) consists of, \( x_{ki} \), \( y_{ki} \), and \( t_{ki} \) denote the 2D location in the \( xy \)-plane and the time coordinate of point \( p_{ki} \), respectively. As usual, we consider linear interpolation between two successive sampled points, \( p_{ki} \) and \( p_{kt+i} \), so that each trajectory consists of a sequence of 3D line segments, \( e_{ki} = [p_{ki}, p_{kt+i}] \), where each segment represents the continuous movement of the object during sampled points.

The problem of sub-trajectory clustering in a MOD is formalized according to the following definition:

**Definition 1:** *Sub-trajectory clustering in a MOD.* Assuming a dataset \( D = \{ T_1, T_2, \ldots, T_N \} \) consisting of \( N \) trajectories, each of which can be considered as a sequence of successive sub-trajectories each of arbitrary length (i.e. \( P_{ki} \) is the \( i \)-th sub-trajectory of trajectory \( T_k \)), the optimal sub-trajectory clustering is the one that partitions the set of sub-trajectories into a clustering \( C = \{ C_1, \ldots, C_M \} \) of \( M \) clusters and a set Out of outliers. Assuming that each cluster \( C_j \) is represented by its representative (or centroid or typical) sub-trajectory \( R_j \), \( j = 1, \ldots, M \), the optimization criterion is to maximize the following expression, that gives the sum of representativeness of the entire dataset (SRD) using the set \( S = \{ R_1, \ldots, R_M \} \) of the representative sub-trajectories and the appropriate corresponding \( M \) clusters \( C(R_j) \). The term \( V(P_{ki}, R_j) \) implies the mean similarity (i.e. average number of votes, according to our terminology) that \( P_{ki} \) sub-trajectory has w.r.t. representative sub-trajectory \( R_j \):

\[
SRD = \sum_{R_j \in S} \sum_{P_{ki} \in C(R_j)} V(P_{ki}, R_j)
\]

Note that maximization of the above expression is a hard problem as one first has to define the criterion with which a trajectory is partitioned to sub-trajectories, the technique for selecting the set of the most representative sub-trajectories whose cardinality \( M \) is unknown, to name but a few challenging sub-problems. In Fig. 2 we outline our proposal that provides a solution to the above problem. The proposed S^2T-Clustering algorithm relies on a voting and segmentation process that detects homogenized sub-trajectories in the MOD, then selects the most representative ones to serve as the seeds of the clusters, and around them, forms the clusters (and isolates the outliers).
Algorithm $S^2T$-Clustering

**Input:** dataset $D = \{T_1, T_2, \ldots, T_N\}$, parameter $\varepsilon$

**Output:** Sampling set $S$, Clustering $C$, Set of outliers $Out$.

1. Calculate the set $V$ of voting descriptors in $D$
2. for each trajectory $T_k \in D$ do
3. Partition $T_k$ in a set $P_k$ of sub-trajectories according to its voting descriptor $V_k$
4. Among the resulting sub-trajectories, find the $M$ most representative to form the sampling set $S$
5. Taking $S$ into consideration, partition the sub-trajectories in a set $C$ of $M$ clusters and a set $Out$ of outliers
6. Return $(S, C, Out)$

Fig. 2: The proposed $S^2T$-Clustering Algorithm.

In the following subsections of Section 3, we will formally present our proposal that solves the sub-trajectory clustering problem as defined above, while Section 4 will provide a step-by-step materialization of it. Table 1 summarizes the definitions of the symbols used in this report.

### 3.1 Voting and segmentation into sub-trajectories

We exploit on the concept of “representativeness” of trajectories in MOD as defined in [35], by extending the definition of density biased sampling (DBS) in point sets [23] for trajectory segments. According to DBS, the local density for each point of the set is approximated by the number of points in a region, divided by the volume of the region. In our case, the “representativeness” of a trajectory segment is defined by the number of the objects that follow this segment along with time, space and direction. Representativeness is calculated by a voting process that is applied upon to each segment $e_{k,i}$ of a given trajectory $T_k$. Thus, $e_{k,i}$ receives votes by each trajectory in MOD, according to their mutual distance. The sum of these votes indicates the number of trajectories being (spatio-temporally) similar to $e_{k,i}$, with the total number $N$ of trajectories serving as upper bound. Under this definition, voting has the physical meaning of how many objects co-move (i.e. are co-located and co-exist) for a period of time. Thus, the voting results can then be post-processed in order for the representative sub-trajectories to be identified.

Let $V_k$ be the voting trajectory descriptor along the $T_k$ line segments, consisting of $L_k - 1$ components. Each component $V_{k,i}$ of this vector corresponds to the number of votes (“representativeness” value) segment $e_{k,i}$ received, $i \in \{1, 2, \ldots, L_k - 1\}$.

This representativeness value is based on a distance function between two 3-D line segments $d(e_{k,i}, e_j)$. The distance function is defined as the definite integral of the time-varying distance $D_j$ between the two segments during their common lifespan $[t_{j,\text{start}}, t_{j,\text{end}}]$, following the approach proposed by Frentzos et al. in [10].

\[
d(e_{k,i}, e_j) = \int_{t_{j,\text{start}}}^{t_{j,\text{end}}} D_j dt \tag{2}\]

As $D_j$ follows a trinomial and as proved in [10] its integral can be computed in $O(1)$ by distinguishing between two cases for the value of the non-negative factor $a (a = 0$ and $a > 0)$. This integral can be
efficiently approximated by the Trapezoid Rule

\[
\frac{D_j(t_{j-start}) + D_j(t_{j-end})}{2} \cdot (t_{j-end} - t_{j-start})
\]

, providing also bounds for the approximation error. Given the above distance function, the representativeness value is provided by the following voting function.

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

Table 1. Table of symbols

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>A dataset, (D = {T_1, T_2, \ldots, T_N}), consisting of (N) trajectories</td>
</tr>
<tr>
<td>N</td>
<td>The cardinality of (D)</td>
</tr>
<tr>
<td>(T_k)</td>
<td>(k)-th trajectory of (D)</td>
</tr>
<tr>
<td>(L_k)</td>
<td>Number of points forming trajectory (T_k)</td>
</tr>
<tr>
<td>(p_{k,i})</td>
<td>(i)-th (3D) point of trajectory (T_k), (p_{k,i} = (x_{k,i}, y_{k,i}, t_{k,i}))</td>
</tr>
<tr>
<td>(e_{k,i})</td>
<td>(i)-th (3D) line segment of trajectory (T_k), (e_{k,i} = [p_{k,i}, p_{k,i+1}])</td>
</tr>
<tr>
<td>(L_{k,i})</td>
<td>Lifespan of (e_{k,i}), calculated as: (L_{k,i} = t_{k,i+1} - t_{k,i})</td>
</tr>
<tr>
<td>(LP_k)</td>
<td>Number of sub-trajectories partitioning trajectory (T_k)</td>
</tr>
<tr>
<td>(P_k)</td>
<td>Set of the sub-trajectories partitioning trajectory (T_k)</td>
</tr>
<tr>
<td>(P_{k,i})</td>
<td>(i)-th sub-trajectory of trajectory (T_k)</td>
</tr>
<tr>
<td>(V)</td>
<td>Set of all voting descriptors in dataset (D)</td>
</tr>
<tr>
<td>(V_k)</td>
<td>The voting descriptor of trajectory (T_k)</td>
</tr>
<tr>
<td>(VP_{k,i})</td>
<td>The voting descriptor of sub-trajectory (P_{k,i})</td>
</tr>
<tr>
<td>(Nl_{k,i})</td>
<td>The descriptor of sub-trajectory (P_{k,i}) w.r.t. normalized lifespan of its line segments</td>
</tr>
<tr>
<td>(S)</td>
<td>Sampling set of representatives, (S = {R_1, \ldots, R_M}), consisting of (M) sub-trajectories</td>
</tr>
<tr>
<td>(M)</td>
<td>The cardinality of (S), also the number of clusters in the resulting clustering</td>
</tr>
<tr>
<td>(SR(S))</td>
<td>Representativeness function of (S)</td>
</tr>
<tr>
<td>(V(P_{k,i}, P_{m,n}))</td>
<td>Voting descriptor of (P_{k,i} \in D-S) w.r.t. (P_{m,n} \in S)</td>
</tr>
<tr>
<td>(C)</td>
<td>Clustering of sub-trajectories in (M) clusters, (C = {C_1, \ldots, C_M})</td>
</tr>
<tr>
<td>(Out)</td>
<td>Set of sub-trajectories not belonging to (C) (i.e., outliers)</td>
</tr>
</tbody>
</table>

The control parameter \(\sigma > 0\) shows how fast the function (“voting influence”) decreases with distance. It holds that \(0 \leq V(e_{k,i}, e_j) \leq 1\). If \(d(e_{k,i}, e_j)\) is close to zero, the voting function gets its maximum value, i.e. 1. This means, that \(e_j\) is (in time, space and direction) very close to \(e_{k,i}\). Otherwise, if \(d(e_{k,i}, e_j)\) is high, the voting function results in almost 0, meaning that \(e_j\) is very far away from \(e_{k,i}\).

After voting, the trajectory segmentation process follows. The goal of this step is to partition each trajectory into sub-trajectories of homogenous representativeness, irrespectively of their shape complexity. Let \(P_{k,i}, i \in \{1, \ldots, LP_k\}\) be the \(i\)-th sub-trajectory of \(T_k\), where \(LP_k\) denotes the number of partitions of \(T_k\). Then, \(VP_{k,i}\) is the vector (descriptor) of votes along the line segments of partition (sub-trajectory) \(P_{k,i}\), showing how many objects find themselves to be similar to \(P_{k,i}\). This value will turn out to be useful for clustering purposes, as it will be presented in the section that follows.

Coming back to the example of Fig. 1, the above discussion will result in partitioning of trajectory \(T_1\) into three sub-trajectories (see red, blue, black portions) according to its representativeness.

### 3.2 Sub-trajectory sampling, clustering and outlier detection

Trajectory segmentation aims to provide homogenous sub-trajectories concerning their representativeness. On the other hand, the goal of sub-trajectory clustering is to partition the dataset into groups (clusters) and to detect the outliers among the sub-trajectories identified in the trajectory segmentation step. Therefore, in our proposal, first we solve the problem of selecting the sampling set \(S\) and then, based upon the former, we tackle the problem of clustering according to the following idea:
each sub-trajectory of the sampling set is considered as a cluster representative. So, our goal is that the sampling set should contain high voted trajectories of the MOD and, at the same time, should cover the entire 3D space as much as possible in order for Equation (1) to be maximized. We propose the sampling to be done by minimizing a formula (see Equation (4)), taking into account the votes (i.e. representativeness) \( V_{P_{k,i}} \) in the original MOD as well as the votes \( V_{P_{k,i}} \) in the sampling MOD and the vector of the lifespans of the line segments divided by the lifespan of the trajectory \( N_{l_{k,i}} \). \( V_{P_{k,i}} \) is computed in the sampling MOD using the voting method similar to \( V_{P_{k,i}} \) computation in the original MOD. Thus, it takes into account the fact that the sampling set that maximizes the optimization formula should contain representative sub-trajectories that would be quite dissimilar to each other.

Let \( S \) denote the sampling set, so that \( S_{k,i} \) is one, if sub-trajectory \( P_{k,i} \) belongs to the sampling set, and zero otherwise. According to the previous properties, the number of moving objects of the original MOD that are represented in sampling set \( SR(S) \), should be maximized. This is formalized in Equations (4) and (5).

\[
SR(S) = \sum_{k=1}^{N} \sum_{i=1}^{L} S_{k,i} \cdot SR_{gain}(k,i) 
\]

where

\[
SR_{gain}(k,i) = \sum_{j=1}^{[P_{k,i}]} VP_{k,i,j} \cdot N_{l_{k,i,j}} \cdot (1 - VP_{k,i,j}) 
\]

\( SR_{gain}(k,i) \) expresses the gain of \( SR(S) \) if we add \( P_{k,i} \) in \( S \). \([P_{k,i}]\) denotes the number of line segments of \( P_{k,i} \). The values \( VP_{k,i,j} \), \( N_{l_{k,i,j}} \) and \( VP_{k,i,j} \) denote for the votes in \( D \), the normalized lifespan and the votes in \( S \) of the \( j \)th line segment of \( P_{k,i} \), respectively.

Then, the clustering is done by minimizing a formula, taking into account the sampling set \( S \) and the vector of votes (i.e. representativeness) \( V(P_{k,i}, R_j) \) between, on the one hand, sub-trajectories of the original MOD \( P_{k,i} \in D - S \) and, on the other hand, the representative sub-trajectories \( R_j \in S \). \( V(P_{k,i}, R_j) \) consists of \([P_{k,i}]\) elements, where each represents the voting that takes place between the corresponding line segments of \( P_{k,i} \) and \( R_j \) sub-trajectories. According to Equation (1), we have used the mean value of the vector values \( V(P_{k,i}, R_j) \), i.e. \( \bar{V}(P_{k,i}, R_j) \). Each value of \( V(P_{k,i}, R_j) \) is computed by measuring the distance of the corresponding line segment of \( P_{k,i} \) from its nearest in \( R_j \) and then by applying the voting function of Equation (3). As such, \( 0 \leq \bar{V}(P_{k,i}, R_j) \leq 1 \).

Therefore, in order to maximize Equation (1), the cluster \( C(R_j) \) of a sub-trajectory of the sampling dataset \( R_j \in S \), i.e. the number of sub-trajectories that are assigned to cluster \( C(R_j) \), is provided by:

\[
C(R_j) = \{ P_{k,i} \in D - S : \bar{V}(P_{k,i}, R_j) \geq \bar{V}(P_{k,i}, R_j), \forall R_j \in S \wedge \bar{V}(P_{k,i}, R_j) \geq \varepsilon \}
\]

On the other hand, the sub-trajectories that are considered outliers (thus forming the outliers set \( Out \)) are defined by the subset of sub-trajectories of the original MOD that are not assigned to any of clusters, as such satisfying the following property:

\[
Out = \{ P_{k,i} \in D - S : \bar{V}(P_{k,i}, R_j) < \varepsilon, \forall R_j \in S \}
\]

Parameter \( \varepsilon \) is a positive real number close to zero that acts as a lower bound threshold of similarity between sub-trajectories and representatives. As such it controls the size of outliers \( Out \).

4. THE PROPOSED FRAMEWORK

In this section, the proposed four-step framework that materializes our proposal is presented. It consists of effective solutions to the problems of (i) trajectory voting and segmentation, (ii) sub-trajectory sampling, and (iii) sub-trajectory clustering and outlier detection.
Fig. 2 illustrates the flow of the proposed framework. In the subsequent sections we present the components that materialize our proposal.

![Diagram of the proposed framework](image)

**Figure 2: Scheme of the proposed framework.**

**Step 1 & 2: trajectory voting and segmentation**

To perform voting-based trajectory segmentation, we adopt and appropriately adapt the *Global Voting Algorithm* (GVA) and the *Trajectory Segmentation Algorithm* (TSA), both proposed in [35]. The input of GVA is a MOD $D = \{T_1, T_2, \ldots, T_N\}$ indexed by a R-tree-like structure such as the TB-tree [40] or the 3D-R-tree [42], as described in [11], a trajectory $T_k \in D$ and the $\sigma > 0$ parameter. The output of GVA is a vector $V_k$ consisting of $L_k - 1$ components, which can be considered as a trajectory descriptor along the line segments $e_k(i), i \in \{1,2,\ldots,L_k-1\}$, of trajectory $T_k$. As such, the components of vector $V_k(i)$ correspond to the voting (representativeness) received by the 3D line segments for each $e_k(i)$ of $T_k$. More details about the implementation of this method are given in [35].

The result of the voting process is the input to the trajectory segmentation algorithm TSA. The input of TSA is the normalized trajectory voting signal $V_k$, and two intrinsic parameters $w, \tau$. The normalization is done by dividing $V_k$ by the maximum of $V_k$, thus bounding $V_k \leq 1$. Parameter $w$ sets the minimum size of a partitioned trajectory; as such it expresses the minimum number of line segments that can define a sub-trajectory. Parameter $\tau$ is related with the segmentation sensitivity of our method. As $\tau$ increases, the number of sub-trajectories reduces. It holds that $\tau$ can be set as a positive number close to zero (e.g. 0.01) [35]. The output of the method is the segmentation $P_k$ of $T_k$ into $LP_k$ partitions, where $LP_k$ is automatically estimated by the proposed scheme.

**Step 3: sub-trajectory sampling**

Having segmented the trajectories of a MOD according the above (voting-and-segmentation) process, we perform sampling in order to select the top-$M$ representative sub-trajectories. For this purpose, the *Sub-trajectory Sampling Algorithm* (SSA), described in [35], is adopted. The input of SSA is the set of sub-trajectories $P_k$ as segmented by TSA, the voting $VP_{k,i}$ and the normalized lifespan $Nl_{k,i}$ vectors of these sub-trajectories. The output of SSA is the sub-trajectory sampling set $S$ consisting of $M$ samples. It is worth to note that $M$ is not user-defined; in contrast, it is dynamically estimated by SSA. As such, SSA provides a deterministic solution, contrary to other probabilistic techniques [23][32], which provide a randomly constructed sampling set trying to fit it to a desired distribution, or user-supervised, explorative approaches [2][3] that base on clustering, which in turn depends on the clustering algorithm itself and its tuning.

**Step 4: sub-trajectory clustering and outlier detection**

SSA results are used to define the clusters. To this end, we propose the so-called *Optimal Sub-trajectory Clustering Algorithm* (SCA$^O$) that is presented in the following paragraphs. The algorithm is called Optimal since it is based on a sampling process that selects the best candidates via an optimization process.
Let $S$ denote the sampling set, so that $S_{k,i}$ is one, if $P_{k,i}$ sub-trajectory belongs to the sampling set, and zero otherwise. The input of SCA$^O$ are the segmented sub-trajectories $P_{k,i}, \forall \ k \in \{1, ..., N\}, i \in \{1, ..., LP_k\}$, the sampling set $S$ of SSA, and the parameter $\epsilon$. The output of the method is the clustering $C$ and the outlier set $Out$ of sub-trajectories. The pseudocode of the proposed Sub-trajectory Clustering Algorithm (SCA$^O$) is given in Figure 3.

First, each cluster is initialized by a sub-trajectory from the sampling set. Each sub-trajectory of the sampling set constitutes the first member (seed) of the corresponding cluster ($C_n(1), n \in \{1, .., M\}$). Then, we apply a two-step filtering procedure so as to increase the efficiency of the algorithm in terms of execution time. At the first step, for each cluster seed $R_j$ we apply an INN query which returns the sub-trajectories $P_{k,i}$ that are “close” to the cluster seed and don’t belong to the sampling set $S$. Subsequently, for each segment $e_{k,i}$ that belong to these sub-trajectories $P_{k,i}$ we apply again the INN query, in the effort to identify the segments of the representative sub-trajectories $R_j$ that are “close” to them and we calculate the corresponding voting. Then, for each sub-trajectory $P_{k,i}$ that does not belong in the sampling set $S$ we calculate its average voting. By taking into account the parameter $\epsilon$ discussed earlier we assign it to the specified cluster and remove it from the outlier set $Out$. In case a sub-trajectory $P_{k,i}$ is returned as a result to more than one cluster seed INN queries, then we assign it to the one that has the highest voting.

More specifically, the algorithm starts by initializing the outlier set $Out$ to contain all the sub-trajectories except the ones that belong to the sampling set $S$ (line 2). Moreover, the voting of every sub-trajectory is set to zero (line 3) and $m$, which indicates the cluster id, is set to 1 (line 1). Afterwards, every sub-trajectory is assigned to be the first element of each cluster, thus acting as the cluster seed (lines 4-7). Finally, the aforementioned two-step filtering process and voting are performed (lines 8 - 21).

**Algorithm SCA$^O$**

**Input:** Sampling set $S$, set of sub-trajectories $P_{k,i}$, parameter $\epsilon$.

**Output:** Clustering $C$ of clusters $C_n, n \in \{1, ..., M\}$, Outliers $Out$.

```
1 m=1
2 Out = P - S
3 V(P_k,i) = 0
4 for each $R_j \in S$ do
5    $C_m(1) = R_j$
6    m=m+1
7 end
8 for each $R_j$ do
9    for each $e(P_{k,i}) \in P_{k,i} \leftarrow$ INN($R_j, P_{k,i}$) do
10     if $V(e(P_{k,i}), e(R_j)) > V(P_{k,i})$ then
11        $V(P_{k,i}) = V(e(P_{k,i}), e(R_j))$
12 end
13 if $V(P_{k,i}) > \epsilon$ then
14     $C_m = C_m \cup P_{k,i}$
15 end
16 Out = Out - P_{k,i}
17 end
```

Figure 3: Optimal Sub-trajectory Clustering Algorithm (SCA$^O$).

### 4.1 A Constraint-aware Sub-trajectory Clustering Algorithm

The S$^T$-Clustering algorithm so far solves the problem of sub-trajectory algorithm as an optimization problem. However, as already discussed, real-world applications may impose additional constraints, which are contradicting with the method used that optimally solves the problem. As in our case optimization takes place at the sampling step, in order to provide a constraint-aware algorithm we need to...
revisit the sampling process and prune out of the clustering step those candidates sub-trajectories (i.e. representatives around which clusters are formed), which cannot satisfy the initially posed constraints. To succeed this we unify in one step the sampling and clustering steps in one (as implied by the dotted line in Figure 2). More specifically, we adapt the SSA algorithm so as in each step to select the optimal representative sub-trajectory, however this representative in order to be added to the sampling set $S$, it should first satisfy the constraints of the group of sub-trajectories which is formulated according to the SCA$^0$ specifications. Otherwise, this initially selected sub-trajectory is ignored and we search for the next candidate. Of course, this way we may scan the entire set of sub-trajectories without being able to identify enough clusters, meaning that we result in a huge set of outliers. In such a case we could relax the specifications for a sub-trajectory to belong to a cluster, by relaxing the parameter $e$ of Equation (6). Obviously, each scan of the remaining set of sub-trajectories re-examines sub-trajectories that have failed in a previous scan. This also implies that each scan results in a set of clusters belonging to an equivalence class of certain parameters. The above described process is presented in detail in Figure 4. As such, the so-called Constraint-aware Sub-trajectory Clustering Algorithm (SCA$^C$) is able to control the size of the set of outliers, while at the same time satisfying any user-defined constraints for the resulted clusters. Of course, an implicit outcome of the algorithm is a different sampling set $S$, which now consists of the representatives around which can be formed clusters satisfying the placed constraints.

Algorithm SCA$^C$

Input: set of sub-trajectories $P_{k,i}$, and set $V$ of their voting descriptors $VP_{k,i}$, parameter $\varepsilon$, maximum allowable number of outliers $Out_{\text{max}}$.

Output: Sampling set of representatives $S$, clustering $C$ of clusters $C_{i,e}$, $i \in \{1, ..., M\}$ and $ec \geq 1$, Outliers $Out$.

1. $m = 0$; $ec = 1$; $e_{ec} = \varepsilon$
2. $C = \emptyset$; $Out = P_{k,i}$
3. $V = \text{sort\_descending}(V)$
4. while $|C| \leq M \text{ AND } |Out| > Out_{\text{max}}$
5. $SR_{\text{gain}}^\text{max} = -1$
6. for each $P_{k,i}$ do
7. if $S_{k,i} = 0$ AND $SR_{\text{gain}}^\text{(k, i)} > SR_{\text{gain}}^\text{max}$
8. then
9. $SR_{\text{gain}}^\text{max} = SR_{\text{gain}}^\text{(k, i)}$
10. $t = k$
11. $g = i$
12. end
13. if $SR_{\text{gain}}^\text{max} \geq \overline{VP_{t,g}}$ then
14. $(t_C, t_{Out}) = \text{SCA}^0(P_{t,g}, Out, e_{ec})$
15. if $\text{constraints\_satisfied}^\text{(t_C)}$ then
16. $m = m + 1$
17. $C_{m,ec} = t_C$
18. $S_{t,g} = 1$
19. $C = C \cup C_{m,ec}$
20. $Out = Out \setminus t_C$
21. else
22. continue
23. end
24. end
25. $ec = ec + 1$; $e_{ec} = e_{ec} \cdot \delta$ \quad // where $\delta \leq 1$,
e.g. $\delta = 0.9$
26. end

Figure 4: Constraint-aware Sub-trajectory Clustering Algorithm (SCA$^C$).
4.2 A note on complexity analysis

Concerning the computation cost, the time complexity of GVA is $O(\bar{L} \cdot N)$, where $\bar{L}$ denotes the mean number of trajectory points [35]. Executing GVA for each trajectory of the database, the total computation cost is $O(\bar{L} \cdot N \cdot \log(\bar{L} \cdot N))$ since $D$ is indexed by an R-tree-like structure. The complexity of TSA is $O(\bar{L} \cdot N)$ while that of SSA is $O(N \cdot \frac{\bar{L}}{LP} \cdot \log(N \cdot \frac{\bar{L}}{LP}) + \bar{L} \cdot M \cdot N \cdot \log(\bar{L} \cdot M))$ [35], where $\frac{\bar{L}}{LP}$ denotes the mean number of trajectory segments and $\frac{\bar{L}}{LP}$ denotes the mean number of sub-trajectory points.

Concerning SCA$\text{^0}$, the initialization step (lines 3-6) requires $O(M)$ steps. In the worst case, the main part of SCA is $O(N' \cdot M)$, where $N'$ denotes the number of sub-trajectories resulted by the segmentation of the $N$ trajectories. As $M << N'$ the complexity of SCA is $O(N')$. Overall, the most computationally intensive part of the proposed method is SSA, thus making the overall complexity of the proposed S$^2$T-Clustering be $O(N \cdot \frac{\bar{L}}{LP} \cdot \log(N \cdot \frac{\bar{L}}{LP}) + \bar{L} \cdot M \cdot N \cdot \log(\bar{L} \cdot M))$.

4.3 S$^2$T-Clustering as an efficient query operator in a MOD

The previous complexity analysis implies that an efficient solution is necessary. In [35] we showed that an efficient implementation of the demanding sampling process requires the use of an incremental nearest neighbor (INN) algorithm, while the MOD is indexed by an R-tree-like structure [11]. However, given the specifications we posed in the introduction of the report, this choice is not applicable due to that the access methods supported by real ORDBMS either do not allow the implementation of an INN algorithm, or their indexing extensibility interface has the discussed shortcomings. This implies that in our case we are forced to use classic NN queries (or range queries simulating NN ones by post-processing).

An additional problem on top of the previous is that although access methods on mobility data is a well-studied domain, indexing mechanisms in existing ORDBMS still lack a unified structure for space and time dimensions for mobility data that satisfy the specifications described earlier. In this report we introduce pg3D-Rtree, an indexing mechanism appropriately designed for mobility data, developed on top of the GiST interface [19] available on PostgreSQL that fulfills our requirements. GiST was inspired by Rtree [17] to provide an interface to efficiently index text, spatial and practically any kind of data in a tree structure. Pg3D-RTree implements the six template functions as specified by GiST [19], so that mobility data can be indexed in a unified 3D-Rtree-like [42] structure. More specifically, pg3D-RTree is designed to operate on 3D line segments $e_{k,i}$. Essentially, the leaf entries in the tree structure contain the actual segments, so a single lookup in the index is sufficient enough to retrieve the actual answer set. The interested reader is referred to [44] for a technical discussion regarding the customization of the GiST’s functions that define our indexing mechanism for mobility data types. Similarly to the pg3D-Rtree index, there exists the N-D indexing mechanism of PostGIS which supports 3D and 4D geometries. Note that the index leaf entries are MBBs (instead of segments), hence we need to visit the respective table page in order to retrieve the actual answer set. This mechanism can be utilized in order to simulate the pg3D-Rtree index.

The proposed algorithms were implemented on top of Postgresql using two different approaches. In the first approach an extension of Postgresql that we have implemented for storing, indexing (pg3D-Rtree) and querying spatiotemporal data, called Hermes, is utilized. In the second approach we employ the PostGIS extension. Note that in this approach a segment of a trajectory is modeled as a 3D linestring and is indexed by the 3D indexing mechanism of PostGIS. More specifically, we maintain 3 relations for storing the trajectories, the sub-trajectories and the segments. Each tuple of the trajectory relation, stores...
information about each trajectory’s id and the trajectory itself in two forms, as a 3D linestring and a spatiotemporal trajectory in correspondence to the PostGIS and the HERMES implementation respectively. The sub-trajectory relation stores the id of each sub-trajectory, a boolean field which indicates whether it is a representative or not, the SRgain_zero measure defined in [35], the cluster in which it belongs and a metric indicating the fitness of the assignment of the sub-trajectory to the specific cluster. Finally, the segment relation contains tuples which contain information about each segment’s id and the actual segment, again, in two forms, as a 3D linestring and a spatiotemporal segment. Additionally, it also stores the normalized voting and normalized lifespan measures described in [35].

At this point, before we proceed with the implementation details, it would be wise to discuss the way that the incremental nearest neighbor (INN) algorithm is simulated. For this reason we propose the Trajectory Buffer Query (TBQ). The TBQ takes as input a trajectory, calculates its trajectory buffer and returns the segments that overlap with it. Before defining formally the TBQ, it is necessary to define what a trajectory buffer is. The trajectory buffer, similarly to a trajectory, is a dynamic data type which, instead of segments, consists of the MBBs of the segments expanded by a spatial distance and a temporal interval. Formally, let us assume that $T$ is a trajectory (composed of a sequence of segments), $\sigma$ is a spatial distance and $\delta$ is a temporal interval. We define $\text{TB}(T, \sigma, \delta)$ to be a 3D ‘buffer’ around $T$ with the property that every point in $\text{TB}(T, \sigma, \delta)$ is at most $\sigma$ and $\delta$ (in space and time, resp.) far from a point in $T$. Hence, given a set $S$ of trajectories, a reference trajectory $T$, a spatial threshold $\sigma$ and a temporal interval $\delta$, the trajectory buffer query $\text{TBQ}(S, T)$ retrieves those sub-trajectories in $S$ having the property that the segments they are composed of intersect with segments $T$ composes of. More specifically, having implemented the trajectory buffer as a data type of the HERMES extension of Postgresql we modify the aforementioned pg3D-RTree index. This modification aims to providing the ability to the pg3D-RTree to be traversed by the trajectory buffer data type when performing an overlap query. To achieve this, it is sufficient enough to modify the consistent method of the existing 3D-RTree GiST index. More particularly,

$$\text{Consistent}(E, q):$$ given an entry $E = (p, \text{ptr})$ (in our case the spatiotemporal segments that constitute the tree) and a query predicate $q$ (in our case the Trajectory Buffer), this function returns false if $p \& \& q$ is certain to be unsatisfiable.

As a result, the 3D-RTree, which is built upon segments, takes as a predicate a trajectory buffer and returns false if a segment is guaranteed that does not overlap with it and true otherwise. Oversimplifying the above, it returns all the segments that overlap the specified trajectory buffer that was given as input by utilizing the existing 3D-RTree index. Furthermore, since the trajectory buffer query is not available in PostGIS, we tried to simulate it in two different ways. The first effort of simulating this query was by calculating the MBB of each segment. The second way of simulating the trajectory buffer query was by calculating the MBB of the entire trajectory. In both cases, the calculated MBBs were afterwards expanded by the user defined spatial and temporal distances of the buffer.

Having the TBQ in mind, it would be interesting to extend it in a way that it could simulate the GVA Algorithm presented in [35]. More specifically, for every trajectory the aforementioned TBQ is posed, thus simulating the NN Query presented in the GVA Algorithm. The spatial threshold $\sigma$ and the temporal interval $\delta$ of the TBQ are given as input by the user. Up to this point, the TBQ is used as it is. The basic idea is to be able to calculate the voting of each segment of a trajectory while its trajectory buffer traverses the pg3D-RTree. More specifically, a new memory context is defined which will allow data to “live” as long as a single tree search lasts. In this memory context, a pointer is maintained, which by definition is similar to an array and can be accessed as one. While traversing the pg3D-RTree for every trajectory buffer, the voting descriptor of every segment of that trajectory buffer is stored contiguously in memory with the help of the aforementioned pointer. The voting descriptor can, then, be accessed by using the subscript of its position which, in fact, is also the segment id. Whenever a leaf entry is reached
by a trajectory buffer that traverses the pg3D-RTree we iterate through the boxes of the specific trajectory buffer. If a box overlaps with the segment existent at the leaf entry, the voting between this segment and the segment contained in the box of the trajectory buffer is calculated and it’s appended in the appropriate position in memory. Finally, after each trajectory buffer finishes traversing the 3D-RTree and the final voting descriptors are materialized into the segment database relation in order to be able to use these voting descriptors in the next steps of S2T Clustering and to able to free the memory for the next trajectory buffer that will traverse the 3D-RTree. Consequently, the normalized voting and the normalized lifespan are computed and stored and the corresponding fields in the segment relation.

Furthermore, since the trajectory buffer and consequently the TBQ are not available in PostGIS, we tried to simulate it in two different ways. The first effort of simulating the trajectory buffer was by calculating the MBB of each segment. The second way of simulating it was by calculating the MBB of the entire trajectory. In both cases, the calculated MBBs were afterwards expanded by the user defined spatial and temporal distance of the buffer. Since we have simulated the trajectory buffer, in order to pose the TBQ it is sufficient enough to overlap the trajectory buffer with the rest of trajectories by using the 3D indexing mechanism of PostGIS. Likewise GVA, the TSA algorithm is also applicable per trajectory. The segment relation is employed and more specifically the voting signal of each trajectory. After the segmentation process is performed, the sub-trajectory relation is updated with the produced sub-trajectories.

The SSA algorithm, as described in [35], takes as input the set of sub-trajectories as estimated by TSA, the voting of the sub-trajectories and the normalized lifespan of the trajectory segments as calculated by GVA. As observed, only the voting of the sub-trajectories is not already computed. Therefore, a voting procedure has to be executed again, but this time for the sub-trajectories. This voting procedure has to be executed between a sub-trajectory that is not in the sampling set and the sub-trajectories that are in the sampling set. Assuming that the number of sub-trajectories existent in the sampling set is small, since they are intricately linked with the number of clusters, then it is obvious that the index is not going to be utilized by the RDBMS. Thus, there is no point in employing the variation of the TBQ, described above, that simulates this voting process. For this reason, in this case, a simple spatiotemporal range query is used in order to simulate the NN query. Subsequently, it is trivial to calculate the voting from the time that all the appropriate sub-trajectories are retrieved. Since the voting of each sub-trajectory is calculated, the SSA algorithm is executed and whenever a new representative sub-trajectory is discovered the sub-trajectory relation is updated setting the boolean field which indicates whether it is a representative or not to true and the cluster id field with the appropriate cluster id.

Finally, the SCA algorithm, describe in section 4, takes as input the set of sub-trajectories $P_k$, produced by TSA and the sampling set $S$ generated by the SSA algorithm. Once more, we notice that INN queries along with a voting procedure are existent. For this reason we employ again the TBQ. In more detail, we need the segment relation joined with the sub-trajectory relation for the sub-trajectories that belong in the sampling set and, for a second time, the segment relation joined with the sub-trajectory relation for the sub-trajectories that don’t belong in the sampling set. We apply the TBQ for each representative sub-trajectory in order to get as a result the segments of the sub-trajectories that are not in $S$ that are “close” to it. At this point, it is worth mentioning that the spatial distance and temporal interval of the TBQ are given as input by the user. At the second filtering step, for each of the resulted segments we want to identify the segments of the representative sub-trajectories that are “close” to them and calculate the corresponding voting. Since it is expected that the number of segments of the representative sub-trajectories that are “close” to them is going to be small, the RDBMS will not use the index. Hence, it would be ineffective to use the variation of the TBQ that also calculates the voting. In fact, at this second filtering step, a simple spatiotemporal range query is posed for each of the resulted segments of the first filtering step, having as spatial and temporal predicates the spatial distance and temporal interval of the TBQ posed at the first step. For each of these resulted segments, the voting is calculated with respect to the queried segment. Subsequently, we calculate the average voting for each sub-trajectory that does not
belong in $S$ with respect to the specific representative sub-trajectory. If this voting is greater than $\epsilon$ and the voting that it had before, possibly from another cluster (representative sub-trajectory), then the voting is updated and the sub-trajectory is assigned to the specified cluster.

5. EXPERIMENTAL RESULTS

5.1 Verification of the Ground Truth

Figure 5: The 2-D map of SMOD with the three one-directional and one bidirectional road.

In order to measure the performance of clustering for movements that can be created by vehicles, we have used a synthetic MOD (SMOD). Note that in order to evaluate our approach, well known synthetic trajectory generators [6] cannot be utilized as we would not be aware of the ground truth of the generated trajectories. As such, assume the following points A(0, 0), B(1, 0), C(4, 0) and D(2, 1) be the destination nodes of a simple graph upon which objects move. We assume that the half of the objects are moving with normal speed (2 units per second) and the rest of them are moving with high speed (5 units per second). The objects are moving under the following scenario (rules), for a lifetime of one second.

- There are three one-directional roads (A → B, B → D, D → C) and one bidirectional road (B ↔ C).
- At $t = 0$ sec, all objects of MOD start from (in a real application, very close to) point A. Thus, the first destination of all objects is point B.
- When an object arrives at a destination point, it ends its trajectory with a probability of 15%. Otherwise, it continues with the same speed to the next point according to the road network rules. If there exist more than one possible next point, it decides randomly the next destination.
- A small number of objects (outliers, four in our experiment) randomly moved in space ignoring roads. In addition, the speed of outliers randomly changes.

As an example, if an object moves from A to B, then the next destination could be C or D with the same probability 42.5%, since it ends its trajectory with a probability of 15%. Fig. 5 illustrates the 2-D map of SMOD with the three one-directional roads and one bidirectional road. According to the rules that the generated trajectories should obey, there exists 8 distinct sub-trajectories in the MOD (ground truth). Therefore, there exist 8 clusters in the MOD. Table 2 illustrates the spatial (first column) and temporal (second column) projection of the 8 clusters.
Figure 6: The trajectories of our synthetic MOD (SMOD) with additive noise of SNR = 50 db projected in (a) 2-D spatial space ignoring time dimension and (b) spatiotemporal 3-D space. The trajectories of our synthetic MOD with additive noise of SNR = 30 db projected in (c) 2-D spatial space and (d) spatiotemporal 3-D space. (e) The four outliers of our synthetic MOD with additive noise of SNR = 50 db projected in 2-D spatial space ignoring time dimension. (f) The four outliers of our synthetic MOD with additive noise of SNR = 30 db projected in 2-D spatial space.

Figure 7: Results of clustering method for synthetic MOD with additive noise of (a) SNR = 50 db and (b) SNR = 30 db. The colormap corresponds to the cluster of each sub-trajectory.
According to this synthetic MOD, the possible endpoints of each trajectory partition are points A, B, C, and D (ground truth). The above dataset is ideal for the purposes of experimentation: It consists of a eight predefined clusters with each one being associated with a known representativeness (as ground truth). In addition the set for outliers is well defined.

<table>
<thead>
<tr>
<th>Path</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A→B</td>
<td>[0, 0.2]</td>
</tr>
<tr>
<td>A→B</td>
<td>[0, 0.5]</td>
</tr>
<tr>
<td>B→C</td>
<td>[0.5, 1]</td>
</tr>
<tr>
<td>B→C</td>
<td>[0.2, 0.8]</td>
</tr>
<tr>
<td>B→D</td>
<td>[0.5, 1]</td>
</tr>
<tr>
<td>B→D</td>
<td>[0.2, 0.52]</td>
</tr>
<tr>
<td>C→B</td>
<td>[0.8, 1]</td>
</tr>
<tr>
<td>D→C</td>
<td>[0.52, 1]</td>
</tr>
</tbody>
</table>

Table 2: The eight clusters of the synthetic MOD (ground truth).

In order to measure the stability of our method to noise effects, we have added Gaussian white noise of different Signal to Noise Ratio (SNR) levels, measured in db, to spatial coordinates of synthetic MOD. The synthetic MOD with additive noise of SNR = 50 db and SNR = 30 db projected in 2-D spatial and 3-D spatiotemporal space is illustrated in Fig. 6. According to this synthetic MOD, there exist eight different paths that are defined by four one-directional roads and two bidirectional roads. We have used 400 moving objects with 1033 sub-trajectories. The mean value and the standard deviation for number of line segments per trajectory (Lk) are 89.18 and 24.51, respectively. The proposed GVA/TSA gives between 1303 and 1438 sub-trajectories under 50 db and 30 db of additive noise, respectively.

Next, we present the results of the proposed clustering method, in other words the results of the proposed GVA/TSA/SSA/SCA four-step methodology. Fig. 7 illustrates results of clustering method for synthetic MOD with additive noise of SNR=50 db and SNR=30 db. We have used colormap according to the cluster that the sub-trajectories belong. The black color has been used for outliers. The proposed method is not affected by the trajectories’ shape, yielding high performance results for both the non-straight and straight movements. In the case of SNR = 50db, the clusters all well detected. In the case of SNR = 30db, the first two clusters (see line 1-2 of Table 2) were partitioned into two sub-clusters due to noise effects. The accuracy of the proposed method concerning the outliers detection, (fraction of corrected classified sub-trajectories divided by the total number of sub-trajectories) was 98.2% and 97.1% when SNR=50 db and SNR=30 db, respectively. The four outliers are well detected when SNR=50 db and SNR=30 db. Therefore, it gives high performance results under any case of additive noise.
REFERENCES


[34] Oracle Corp. 1998. All Your Data: The Oracle Extensibility Architecture.