Mining Time Relaxed Gradual Moving Object Clusters

Nhat Hai Phan, Dino Ienco, Pascal Poncelet, Maguelonne Teisseire

Mining Time Relaxed Gradual Moving Object Clusters

Phan Nhat Hai
LIRMM - Univ. Montpellier 2
nhat-hai.phan@teledetection.fr

Dino Ienco
IRSTEA - Montpellier, France
dino.ienco@teledetection.fr

Pascal Poncelet
LIRMM - Univ. Montpellier 2
pascal.poncelet@lirmm.fr

Maguelonne Teisseire
IRSTEA - Montpellier, France
teisseire@teledetection.fr

ABSTRACT

One of the objectives of spatio-temporal data mining is to analyze moving object datasets to exploit interesting patterns. Traditionally, existing methods only focus on an unchanged group of moving objects during a time period. Thus, they cannot capture object moving trends which can be very useful for better understanding the natural moving behavior in various real world applications. In this paper, we present a novel concept of *"time relaxed gradual trajectory pattern"*, denoted real-Gpattern, which captures the object movement tendency. Additionally, we also propose an efficient algorithm, called *ClusterGrowth*, designed to extract the complete set of all interesting maximal real-Gpatterns. Conducted experiments on real and large synthetic datasets demonstrate the effectiveness, parameter sensitiveness and efficiency of our methods.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining, Spatial Databases and GIS*

General Terms

Theory, Algorithms, Experimentation

Keywords

Gradual moving object cluster, gradual trajectories.

1. INTRODUCTION

Nowadays, the use of many electronic devices in real world applications has led to an increasingly large amount of data containing moving object information. One of the objectives of spatio-temporal data mining [1, 4, 6] is to analyze such datasets for interesting patterns which usually are moving object clusters. A moving object cluster can be defined in both spatial and temporal dimensions: (1) a group of moving objects should be geometrically closed to each other, (2) they should be together for at least some number of certain timestamps. In this context, many recent studies have been defined to mine moving object clusters including flocks [1], moving clusters [4], convoy queries [3], closed swarms [6], group patterns [9], periodic patterns [7], etc... The interested readers may refer to [8] where descriptions of the most efficient approaches and patterns are proposed.

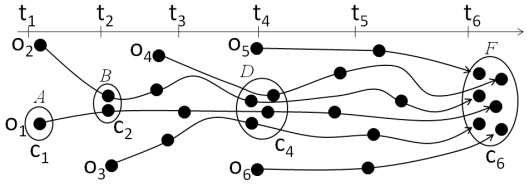


Figure 1: An example of real-Gpattern.

Unfortunately, these patterns cannot help us fully understand the complex object moving behavior. To illustrate, let us consider the Salmon[1] migration in the ocean, where the adult salmons return primarily to their natal stream to spawn. From time to time, more and more salmons get closed together to go to their stream origin. Actually, this phenomenon is involved in many real world applications (e.g. traffic congestion, animal or population migration, etc).

In this paper, we propose a novel movement pattern, *real-Gpattern* (i.e. time relaxed gradual trajectory pattern), which is designed to capture the gradual object moving trend. More precisely, a gradual moving object cluster is a list of moving object clusters satisfying the graduality constraint and integrity condition during at least min_t timestamps. The graduality constraint can be the increase or decrease of the number of objects while the integrity condition can be that all the objects should remain in the next cluster.

For instance, in Figure 1, if we set $min_t = 3$, one object moving trend is *"from t_1 to t_6, the more time passes, the more objects are following the trajectory $\{A \rangle B \rangle D \rangle F\}$"*. Note that moving objects in a cluster may actually diverge temporarily and converge at certain timestamps. Furthermore, if we denote a real-Gpattern as a list of clusters C then, we have 4 patterns: $C_1 = \{c_1, c_2, c_4\}$, $C_2 = \{c_1, c_2, c_6\}$, $C_3 = \{c_2, c_4, c_6\}$ and $C_4 = \{c_1, c_2, c_4, c_6\}$. Actually, they are redundant since C_1, C_2, C_3 are included in C_4.

To avoid finding redundant real-Gpatterns, we further propose the *maximal real-Gpattern* concept. The basic idea is that if C is a real-Gpattern, it is useless to output any subset C' of C. For example, see Figure 1, a maximal real-Gpattern is $C_4 = \{c_1, c_2, c_4, c_6\}$.

Efficient extracting of complete set of maximal real-Gpatterns in a large moving object database, denoted DB, is a non-trivial task: 1) the size of all the possible com-

[1]http://en.wikipedia.org/wiki/Salmon

binations is exponential, 2) none of previous studies (i.e. frequent pattern mining [2], moving object clusters [1, 4, 6]) has addressed the same issue.

Facing the huge potential search space, we propose an efficient approach, named *ClusterGrowth*. In ClusterGrowth, we design two efficient rules which are *Graduality Pruning rule* and *Backward Pruning rule* to avoid unnecessary further search. Additionally, to eliminate uninteresting patterns, we relax the time constraint within a time-based sliding window. Furthermore, we also present an *Actual Maximum Checking* step that reports the *interesting maximal real-Gpatterns* on-the-fly without extra space to store candidates and extra time for post-processing. The effectiveness, parameter sensitiveness and efficiency of our methods are demonstrated on both real and large synthetic databases.

2. PROBLEM STATEMENT

In this section, we give the real-Gpattern and maximal real-Gpattern definitions. Let us assume that we have a set of moving objects $O_{DB} = \{o_1, o_2, \ldots, o_z\}$, a set of timestamps $T_{DB} = \{t_1, t_2, \ldots, t_m\}$, and a gradual variation $* \in \{\geq, \leq\}$.

Database of clusters. A database of clusters, $C_{DB} = \{C_{t_1}, C_{t_2}, \ldots, C_{t_m}\}$, is a collection of snapshots of the moving object clusters at timestamps $\{t_1, t_2, \ldots, t_m\}$. Note that an object could belong to several clusters at one timestamp (overlapping clusters). Given a cluster $c \in C_{DB}$ and $c \subseteq O_{DB}$, $|c|$ and $t(c)$ are respectively used to denote the number of objects belonging to cluster c and the timestamp that c involved in. In this paper, we take clustering as a preprocessing step.

Real-Gpattern and maximal real-Gpattern. A list of clusters $C^* = \{c_1, \ldots, c_n\}$ is said to be a real-Gpattern if each pair of consecutive clusters in C^* satisfies graduality condition and C^* contains at least min_t clusters. The definition of real-Gpattern is as follows.

DEFINITION 1. *Real-Gpattern. Given a list of clusters $C^* = \{c_1, \ldots, c_n\}$ and a minimum threshold min_t. C^* is a real-Gpattern if:*

$$C^* = C^{\geq} : \begin{cases} (1) : |C^*| \geq min_t. \\ \forall i \in \{1, \ldots, n-1\}, \\ (2) : c_i \subseteq c_{i+1}. \\ (3) : |c_n| > |c_1|. \end{cases} \quad (1)$$

$$C^* = C^{\leq} : \begin{cases} (1) : |C^*| \geq min_t. \\ \forall i \in \{1, \ldots, n-1\}, \\ (2) : c_i \supseteq c_{i+1}. \\ (3) : |c_n| < |c_1|. \end{cases} \quad (2)$$

For instance, see Figure 1, there are 6 objects and 6 timestamps: ($O_{DB} = \{o_1, \ldots, o_6\}, T_{DB} = \{t_1, \ldots, t_6\}$). Given $min_t = 3$, $C_1^{\geq} = \{c_1, c_2, c_4\}$ is a real-Gpattern since $|C_1^{\geq}| \geq min_t$, $c_1 \subset c_2 \subset c_4$ and $|c_4| = 4 > |c_1| = 1$. Furthermore, the set of all real-Gpatterns is: $C_1^{\geq} = \{c_1, c_2, c_4\}$, $C_2^{\geq} = \{c_1, c_2, c_6\}$, $C_3^{\geq} = \{c_2, c_4, c_6\}$ and $C_4^{\geq} = \{c_1, c_2, c_4, c_6\}$.

However, it is obviously redundant to output $C_1^{\geq}, C_2^{\geq}, C_3^{\geq}$ since all of them can be enlarged to $C_4^{\geq}$. Therefore, we only focus on extracting maximal real-Gpatterns.

DEFINITION 2. *Maximal Real-Gpattern. Given a real-Gpattern $C^* = \{c_1, \ldots, c_n\}$. C^* is maximal if $\nexists C'^*, C^* \subset C'^*$ and C'^* is a real-Gpattern.*

For instance, in Figure 1, $C_4^{\geq} = \{c_1, c_2, c_4, c_6\}$ is a maximal real-Gpattern.

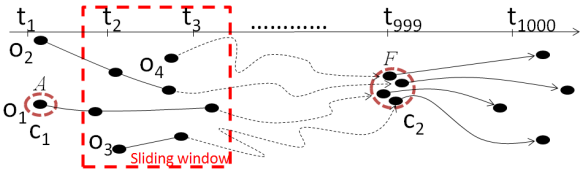


Figure 2: An example of uninteresting real-Gpattern and (time-based) sliding window ($w = 2$).

Table 1: An example of a reconfigured spatio-temporal database in Figure 3.

Timestamp	Object Key	*Cluster*
t_1	1000	c_1
t_2	1100	c_2
t_3	0101	c_3
t_3	1010	c_4
t_4	1111	c_5

Uninteresting real-Gpatterns. As mentioned before the size of all the possible combinations is exponential (i.e. there are totally $2^{|C_{DB}|}$ potential pattern candidates) and naturally not all of them are interesting and useful for analysts. For instance, see Figure 2, $\{c_1, c_2\}$ is a real-Gpattern but it is not interesting and useless. Since the objects o_1, o_2, o_3, o_4 only meet each other at F by chance after 999 timestamps. To eliminate such kind of uninteresting patterns, we propose to relax time constraint within a time-based sliding window with size denoted w. That is, given a current cluster c, c can combine with clusters c' where $1 \leq t(c') - t(c) \leq w$ to be candidates. Note that $t(c') - t(c) \geq 1$ because two clusters at a timestamp cannot belong to a pattern. By using this sliding window, we can ignore a number of uninteresting patterns. For instance, see Figure 2, given that $w = 2$, $\{c_1, c_2\}$ is an uninteresting pattern since $t(c_2) - t(c_1) = 998 > w = 2$.

DEFINITION 3. *Interesting Maximal Real-Gpattern. Given a maximal real-Gpattern $C^* = \{c_1, \ldots, c_n\}$, a time-based sliding window size w. C^* is an interesting pattern if:*

$$\forall i \in \{1, \ldots, n-1\} : 1 \leq t(c_{i+1}) - t(c_i) \leq w \quad (3)$$

3. DISCOVERING MAXIMAL REAL-GRADUAL TRAJECTORY PATTERNS

First we know that the number of different real-Gpatterns could be $2^{|C_{DB}|}$. Second, we need to compute the intersections between clusters and then we have to find an efficient way to manage them. We propose to reconfigure spatio-temporal databases by using a bit set presentation, called object key, for each timestamp and for each cluster as illustrated in Table 1.

Main idea of ClusterGrowth algorithm. For the search space of C_{DB}, we apply a depth-first search on all subsets of C_{DB}, which is illustrated as a pre-order tree traversal in Figure 4: tree nodes are labeled with numbers, denoting the depth-first search order. Note that C_{DB} is ordered by the time from the beginning (resp. t_1) to the end (resp. t_m).

We propose two pruning rules to further shrink the search space. The former, called *Graduality Pruning*, aims at ending traversing the subtree when we find further traversal that cannot satisfy graduality and interestingness requirement (Definition 3). The latter, called *Backward Pruning*, is in charge of managing the maximum property. This rule checks whether there is a superset of the current list of clusters, which has been traversed. If so, the traversal of the subtree under the current list of clusters is meaningless since

all its supersets are not maximal. After pruning the invalid candidates, remaining candidates may or may not be interesting maximal real-Gpatterns. We further propose an *Actual Maximum* checking to embed a maximum checking step in the search process. This checking step immediately determines whether a real-Gpattern C^* is maximal after the subtrees under C^* are traversed. Thus, interesting maximal real-Gpatterns are extracted in the search process and no extra post-processing step is needed.

Now, we formally define some properties for pruning. The ClusterGrowth method is a depth-first-search (DFS) approach based on the cluster set search space.

PROPERTY 1. ***(Graduality Pruning Rule)***. *Given a list of clusters $C^* = \{c_1, c_2, \ldots, c_n\}$, a cluster c' where $t(c') > t(c_n)$ and a window size w. There is no strict superset $C'^* \supseteq (C^* \cup c')$ s.t. C'^* is an interesting maximal real-Gpattern if:*

$$C^* = C^{\geq} : (t(c') - t(c_n) > w) \vee (c' \not\supseteq c_n) \quad (4)$$

$$C^* = C^{\leq} : (t(c') - t(c_n) > w) \vee (c_n \not\supseteq c') \quad (5)$$

In Figure 4, the nodes with list of clusters $C^{\geq} = \{c_1, c_2, c_3\}$ and its subtree are pruned by Graduality Pruning rule because $c_2 \not\subseteq c_3$. This is similar for $\{c_1, c_3\}$, $\{c_1, c_2, c_4\}$.

Even though Graduality Pruning rule can eliminate a large number of useless candidates, there are many other candidates can be pruned. For instance, $\{c_2\}$ subtrees cannot provide any interesting maximal real-Gpatterns. This is because $\{c_1, c_2\}$ has been already traversed and $\{c_1, c_2\}$ is a real-Gpattern. Therefore, for any superset of $\{c_2\}$, denoted $\{c_2\} \cup C^*$, if it is a real-Gpattern then we also have $\{c_1, c_2\} \cup C^*$ is a real-Gpattern. Thus, $\{c_2\} \cup C^*$ is not maximal and therefore $\{c_2\}$ subtrees need to be pruned. The *Backward Pruning* rule can be formalized as follows.

PROPERTY 2. ***(Backward Pruning Rule)***. *Given a list of clusters $C^* = \{c_1, c_2, \ldots, c_n\}$. If there exists a cluster c' such that $t(c') < t(c_n), c' \notin C^*$ and $C'^* = C^* \cup \{c'\}$ satisfies the condition 2 - Definition 1 (i.e. graduality condition) then any supersets of C^* are not interesting maximal real-Gpatterns. Thus, C^* subtrees can be pruned.*

Backward Pruning is efficient in the context since we only need to examine supersets of C^* with one more cluster rather than all the supersets.

After pruning all useless candidates, we need to verify the remaining ones for obtaining the complete set of interesting maximal real-Gpatterns. We can consider that a list of clusters C^* is maximal and interesting if there is no superset of C^*, denoted C'^*, so that C'^* contains at least min_t clusters and the first cluster (resp. $c_1 \in C'^*$) and the last cluster (resp. $c_n \in C'^*$) satisfy the condition 3-Definition 1.

PROPERTY 3. ***(Actual Maximum Rule)***. *Given a list of clusters $C^* = \{c_1, \ldots, c_n\}$. If there exists a cluster c' (i.e. $1 \leq t(c') - t(c_n) \leq w$) so that C'^* is generated by adding c' into C^* and C'^* satisfies the condition 2-Definition 1 then C^* is not an interesting maximal real-Gpattern.*

Note, it is different from the first two rules, this rule does not prune C^* subtrees in the DFS and therefore we cannot end DFS from C^*. However, this rule is useful for detecting non-interesting maximal real-Gpatterns.

THEOREM 1. ***(Interesting maximal real-Gpattern in ClusterGrowth)***. *Given a node with list of clusters $C^* = \{c_1, \ldots, c_n\}$, C^* is an interesting maximal real-Gpattern if and only if it passes all the rules Graduality Pruning, Backward Pruning, Actual Maximum rule, and $|C^*| \geq min_t$ and if c_1, c_n satisfy the condition 3-Definition 1 (i.e. if $* =$ '$\geq$' then $|c_n| > |c_1|$, or if $* =$ '$\leq$' then $|c_n| < |c_1|$).*

Theorem 1 shows that the discovery of interesting maximal real-Gpatterns can be performed without any post-processing step. Figure 4 gives the search space for ClusterGrowth on our running example.

4. EXPERIMENTAL RESULTS

A comprehensive performance study has been conducted on real and synthetic datasets. All the algorithms are implemented in C++, and all the experiments are carried out on a 2.8GHz Intel Core i7 system with 4GB Memory. The system runs Ubuntu 11.10 and g++ version 4.6.1.

The implementation of our proposed algorithm is also integrated in a demonstration system available online[2]. As in [6], *Swainsoni dataset*[3](i.e. 43 objects, 764 timestamps) has been used during experiments. In the comparison, we employ the latest pattern mining algorithms such as *CuTS**[4] [3] (convoy mining) and *ObjectGrowth* [6] (closed swarm mining). Similarly to [3, 6], we first use linear interpolation to fill in the missing data. Furthermore, DBScan [5] ($MinPts = 2, Eps = 0.001$) is applied to generate clusters at each timestamp. To make fair comparison, we adapt all the algorithms to accommodate clusters as input but their time complexity will remain the same. Additionally, the default (resp. hardest) value of min_t is 1, $min_o = 1$ (i.e. for ObjectGrowth and CuTS*). In this paper, w is set to $10\%|T_{DB}|$ for each dataset. It means that $w = 76$ days for Swainsoni dataset, $w =$ 1,000 days for Synthetic dataset.

Effectiveness and Pattern Meaning. The effectiveness of interesting maximal real-Gpatterns can be demonstrated through our online demo system. One of the extracted patterns from Swainsoni dataset is illustrated in Figure 5. Each color represents a Swainsoni trajectory segment involved in the pattern. Additionally, each place mark is a cluster center with the number of objects information and we only report the locations where the #objects changes.

Distinguish from previous work, by proposing interesting maximal real-Gpattern, we are able to capture the moving behavior of the class of Swainsonies in a graduality point of view. Looking at the illustrated pattern in Figure 5, we can consider that they start with 8 objects from the north of America and then they group together to be 11, 14, 16, 20 and 23 objects before flying over the sea. Moreover, during the fly, they continue getting closed each other and at the Colombia, we can observe a total of 27 objects flying together. Interestingly, we also can say that *"from 1996-10-01 to 1996-10-25, the more time passes, the more objects are following the trajectory $\{Oregon\rangle\ Nevada\rangle\ Utah\rangle\ Arizona\rangle\ Mexico\rangle\ Colombia\}$"*. Moreover, on 1996-10-14, they group almost together at some important places such as Mexico where they begin to fly along a narrow corridor through Central America and down to South America. Furthermore,

[2]http://www.lirmm.fr/~phan/realgp.jsp

[3]http://www.movebank.org

[4]The source code of *CuTS** is available at http://lsirpeople.epfl.ch/jeung/source_codes.htm

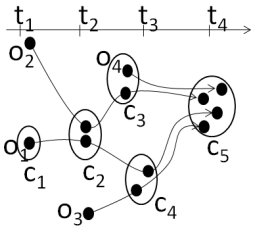


Figure 3: A running example of ClusterGrowth algorithm.

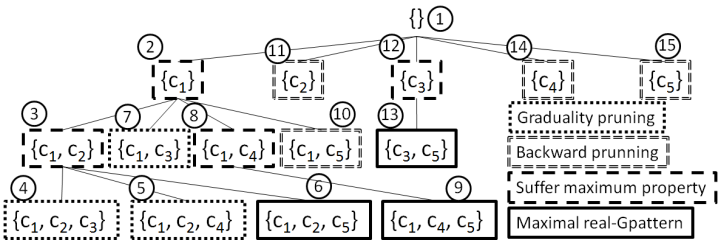


Figure 4: ClusterGrowth search space of the running example in Figure 3 with $* =$ '$\geq$', $min_t = 1$ and $w = 3$.

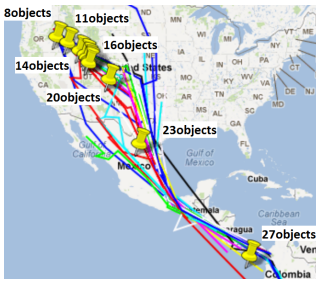


Figure 5: One of extracted real-Gpatterns $C^{\geq}$.

Panama is also important since all objects are together before arriving Colombia, South America.

Parameter Sensitiveness. To show the parameter sensitiveness and efficiency of the proposed algorithm, we also generate a large synthetic dataset using Brinkhoff's network[5]-based generator of moving objects. We generate 500 objects ($|O_{DB}| = 500$) for 10^4 timestamps ($|T_{DB}| = 10^4$) using the generator's default map. There are 5×10^6 points in total. DBScan ($MinPts = 3, Eps = 300$) is applied to obtain clusters at each timestamp.

Sensitiveness w.r.t w**.** See Figure 6a, we can consider that ClusterGrowth is linear in terms of sliding window size w. The reason is that, the higher window sizes w the more patterns are extracted. This is because, for any cluster c, c can combine with an additional number of other clusters corresponding to the increase of w. Consequently, there are more number of candidates and patterns.

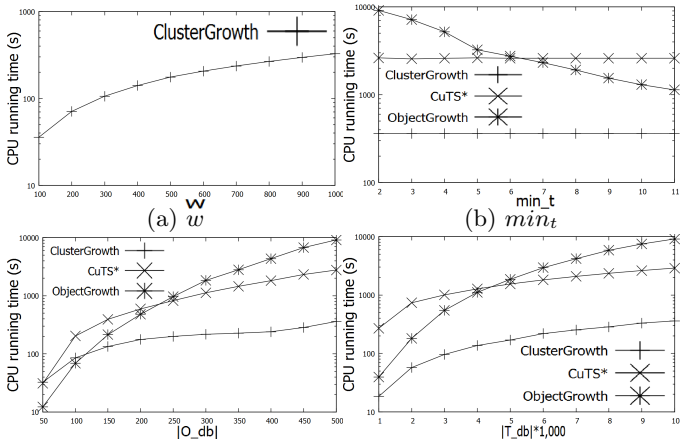
Sensitiveness w.r.t min_t**.** Figure 6b shows that ObjectGrowth is the most sensitive algorithm in min_t. This is because ObjectGrowth applies a min_t-based pruning rule, called *Apriori Pruning*, which is very sensitive in min_t. Since, it is used to limit approximately $2^{|T_{DB}|}$ candidates in total. Furthermore, with different values of min_t, there are great differences in terms of the number of extracted closed swarms. Meanwhile, ClusterGrowth and CuTS* only use min_t at the pattern reporting step without any pruning rule for min_t. Therefore, similar to CuTS*, the ClusterGrowth sensitiveness in min_t is minimized and it is less sensitive than ObjectGrowth.

Sensitiveness w.r.t O_{DB}**,** T_{DB}**.** Once again, ObjectGrowth is the most sensitive algorithm (see Figures 6c-d). The reason is that the number of candidates is greatly increased due to the size increase of $|O_{DB}|, |T_{DB}|$ (i.e. approximately $2^{|O_{DB}|} \times 2^{|T_{DB}|}$ candidates). As the results, the number of closed swarms is significantly increased. Meanwhile, ClusterGrowth and CuTS* do not generate much more candidates as ObjectGrowth does. This is because: 1) the number of clusters at a certain timestamp is not exponentially increased due to the $|O_{DB}|$ and $|T_{DB}|$ increases, 2) for any cluster c, c can combine with the clusters at the next timestamp (i.e. for CuTS*) or the clusters within a sliding window (i.e. for ClusterGrowth). Obviously, ClusterGrowth is similar to CuTS* and less sensitive than ObjectGrowth in terms of $|O_{DB}|$ and $|T_{DB}|$.

5. CONCLUSION

In this paper, we propose the concepts of real-Gpattern and interesting maximal real-Gpattern. These concepts enable the discovery of interesting movement patterns which capture the object moving trends. A novel method, ClusterGrowth is proposed to efficiently discover a complete set of interesting maximal real-Gpatterns. The proposed algorithm effectiveness, efficiency and parameter sensitiveness are demonstrated using real and large synthetic datasets.

[5]http://iapg.jade-hs.de/personen/brinkhoff/generator/



(a) w (b) min_t (c) $|O_{DB}|$ (d) $|T_{DB}|$

Figure 6: Running time on Synthetic Dataset.

6. REFERENCES

[1] J. Gudmundsson, M.van Kreveld. *Computing longest duration flocks in trajectory data.* In GIS '06, pp.35-42.

[2] J. Han, J. Pei, Y. Yin, and R. Mao. *Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach.* In DMKD'04, pp. 53-87.

[3] H. Jeung, ML. Yiu, X. Zhou, CS. Jensen, HT. Shen. *Discovery of Convoys in Trajectory Databases.* PVLDB 2008, 1(1):1068-1080.

[4] P. Kalnis, N. Mamoulis, S. Bakiras. *On Discovering Moving Clusters in Spatio-temporal Data.* In SSTD 2005, Angra dos Reis, Brazil, pages 364-381.

[5] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.* In KDD '96, pp. 226-231.

[6] Z. Li, B. Ding, J. Han, R. Kays. *Swarm: Mining Relaxed Temporal Moving Object Clusters.* In VLDB 2010, Singapore, pp. 723-734.

[7] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, D. W. Cheung. *Mining, Indexing, and Querying Historical Spatiotemporal Data.* In KDD'04, pp.236-245.

[8] V. Bogorny and S. Shekhar. *Spatial and Spatio-Temporal Data Mining.* Tutorial on Spatial and Spatio-Temporal Data Mining, ICDM2010, Australia.

[9] Y. Wang, E.-P. Lim, and S.-Y. Hwang. *Efficient Mining of Group Patterns from User Movement Data.* In DKE '06, pp. 240-282.