GET_MOVE: An Efficient and Unifying Spatio-Temporal Pattern Mining Algorithm for Moving Objects

Phan Nhat Hai, Pascal Poncelet, Maguelonne Teisseire

GeT_Move: An Efficient and Unifying Spatio-Temporal Pattern Mining Algorithm for Moving Objects

Phan Nhat Hai[1,2], Pascal Poncelet[1,2], and Maguelonne Teisseire[1,2]

[1] IRSTEA Montpellier, UMR TETIS - 34093 Montpellier, France
{nhat-hai.phan,maguelonne.teisseire}@teledetection.fr
[2] LIRMM CNRS Montpellier - 34090 Montpellier, France pascal.poncelet@lirmm.fr

Abstract. Recent improvements in positioning technology has led to a much wider availability of massive moving object data. A crucial task is to find the moving objects that travel together. Usually, they are called spatio-temporal patterns. Due to the emergence of many different kinds of spatio-temporal patterns in recent years, different approaches have been proposed to extract them. However, each approach only focuses on mining a specific kind of pattern. In addition to the fact that it is a painstaking task due to the large number of algorithms used to mine and manage patterns, it is also time consuming. Additionally, we have to execute these algorithms again whenever new data are added to the existing database. To address these issues, we first redefine spatio-temporal patterns in the itemset context. Secondly, we propose a unifying approach, named *GeT_Move*, using a frequent closed itemset-based spatio-temporal pattern-mining algorithm to mine and manage different spatio-temporal patterns. GeT_Move is implemented in two versions which are GeT_Move and Incremental GeT_Move. Experiments are performed on real and synthetic datasets and the experimental results show that our approaches are very effective and outperform existing algorithms in terms of efficiency.

Keywords: Spatio-temporal pattern, frequent closed itemset, trajectories

1 Introduction

Nowadays, many electronic devices are used for real world applications. Telemetry attached on wildlife, GPS installed in cars, sensor networks, and mobile phones have enabled the tracking of almost any kind of data and has led to an increasingly large amount of data that contain moving objects. Therefore, analysis on such data to find interesting patterns is attracting increasing attention for applications such as movement pattern analysis, animal behavior study, route planning and vehicle control.

Recently, many spatio-temporal patterns have been proposed [1, 3, 4, 6, 14, 10, 12]. In this paper, we are interested in the querying of patterns which capture *'group'* or *'common'* behaviour among moving entities. This is particularly true to identify groups of moving objects for which a strong relationship and interaction exist within a defined spatial region during a given time duration. Some examples of these patterns are flocks [1, 2], moving clusters [4, 12], convoy queries [3], closed swarms [6, 9], group patterns [14], periodic patterns [10], etc...

Table 1. An example of a Spatio-Temporal Database

Objects O_{DB}	Timesets T_{DB}	x	y
o_1	t_1	2.3	1.2
o_2	t_1	2.1	1
o_1	t_2	10.3	28.1

To extract these kinds of patterns, different algorithms have been proposed. Naturally, the computation is costly and time consuming because we need to execute different algorithms consecutively. However, if we had an algorithm which could extract different kinds of patterns, the computation costs will be significantly decreased and the process would be much less time consuming. Therefore, we need to develop an efficient unifying algorithm. Additionally, in real world applications (e.g. cars), object locations are continuously reported by using Global Positioning System (GPS). Thus, new data is always available. If we do not have an incremental algorithm, we need to execute again and again algorithms on the whole database including existing data and new data to extract patterns. This is of course, cost-prohibitive and time consuming. An incremental algorithm can indeed improve the process by combining the results extracted from the existing data and the new data to obtain the final results.

With the above issues in mind, we propose *GeT_Move*: a unifying incremental spatio-temporal pattern-mining approach. The main idea of the algorithm and the main contributions of this paper are summarized below.

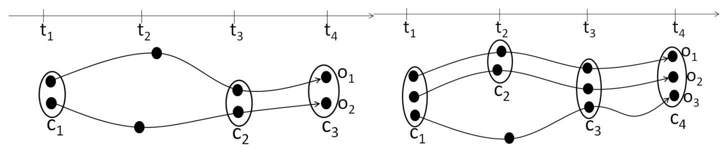
- We re-define the spatio-temporal patterns mining in the itemset context which enable us to effectively extract different kinds of spatio-temporal patterns.
- We present approaches, called *GeT_Move* and *Incremental GeT_Move*, which efficiently extract FCIs from which spatio-temporal patterns are retrieved.
- We present comprehensive experimental results over both real and synthetic databases. The results demonstrate that our techniques enable us to effectively extract different kinds of patterns. Furthermore, our approaches are more efficient compared to other algorithms in most of cases.

The remaining sections are organized as follows. Section 2 discusses preliminary definitions of the spatio-temporal patterns and the related work. The properties of these patterns are provided in an itemset context in Section 3. We introduce the GeT_Move and Incremental GeT_Move algorithms in Section 4. Experiments testing effectiveness and efficiency are shown in Section 5. Finally, we draw our conclusions in Section 6.

2 Spatio-Temporal Patterns

2.1 Preliminary Definitions

The problem of spatio-temporal patterns has been extensively addressed over the last years. Basically, spatio-temporal patterns are designed to group similar trajectories or objects which tend to move together during a time interval. Recently, many patterns have been defined such as *flocks [1, 2], convoys [3], swarms, closed swarms [6, 9], moving clusters [4, 12], group pattern [14]* and even *periodic patterns [10].*



(a) Swarm (b) Convoy

Fig. 1. An example of swarm and convoy where c_1, c_2, c_3, c_4 are clusters which gather closed objects together at specific timestamps.

In this paper, we focus on proposing a unifying approach to effectively and efficiently extract all these different kinds of patterns. First of all, we assume that we have a group of moving objects $O_{DB} = \{o_1, o_2, \ldots, o_z\}$, a set of timestamps $T_{DB} = \{t_1, t_2, \ldots, t_n\}$ and at each timestamp $t_i \in T_{DB}$, spatial information[3] x, y for each object. For example, Table 1 illustrates an example of a spatio-temporal database. Usually, in spatio-temporal mining, we are interested in extracting a group of objects staying together during a period. Therefore, from now, $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_p}\}(O \subseteq O_{DB})$ stands for a group of objects, $T = \{t_{a_1}, t_{a_2}, \ldots, t_{a_m}\}(T \subseteq T_{DB})$ is the set of timestamps within which objects stay together. Let ε be a user-defined threshold standing for a minimum number of objects and min_t a minimum number of timestamps. Thus $|O|$ (resp. $|T|$) must be greater than or equal to ε (resp. min_t). In the following, we formally define all the different kinds of patterns.

Informally, a *swarm* is a group of moving objects O containing at least ε individuals which are closed each other for at least min_t timestamps. To avoid this redundancy, Zhenhui Li et al. [6] propose the notion of *closed swarm* for grouping together both objects and time. A swarm (O, T) is *object-closed* if when fixing T, O cannot be enlarged. Similarly, a swarm (O, T) is *time-closed* if when fixing O, T cannot be enlarged. Finally, a swarm (O, T) is a closed swarm if it is both object-closed and time-closed. Then swarm and closed swarm can be formally defined as follows:

Definition 1 *Swarm and Closed Swarm [6]. A pair (O, T) is a swarm if:*

$$\begin{cases} (1) : \forall t_{a_i} \in T, \exists c \textit{ s.t. } O \subseteq c,\ c \textit{ is a cluster.} \\ (2) : |O| \geq \varepsilon. \\ (3) : |T| \geq min_t. \end{cases} \tag{1}$$

A pair (O, T) is a closed swarm if:

$$\begin{cases} (1) : (O, T) \textit{ is a swarm.} \\ (2) : \nexists O' \textit{ s.t. } (O', T) \textit{ is a swarm and } O \subset O'. \\ (3) : \nexists T' \textit{ s.t. } (O, T') \textit{ is a swarm and } T \subset T'. \end{cases} \tag{2}$$

For example, as shown in Figure 1a, if we set $\varepsilon = 2$ and $min_t = 2$, we can find the following swarms $(\{o_1, o_2\}, \{t_1, t_3\})$, $(\{o_1, o_2\}, \{t_1, t_4\})$, $(\{o_1, o_2\}, \{t_3, t_4\})$, $(\{o_1, o_2\}, \{t_1, t_3, t_4\})$. We can note that these swarms are in fact redundant since they can be grouped together in the following closed swarm $(\{o_1, o_2\}, \{t_1, t_3, t_4\})$.

[3] Spatial information can be for instance GPS location.

A *convoy* is also a group of objects such that these objects are closed each other during at least min_t time points. The main difference between convoy and closed swarm is that convoy lifetimes must be consecutive:

Definition 2 *Convoy [3]. A pair (O,T), is a convoy if:*

$$\begin{cases} (1) : (O,T) \textit{ is a swarm.} \\ (2) : \forall i, 1 \leq i < |T|, \, t_{a_i}, t_{a_{i+1}} \textit{ are consecutive.} \end{cases} \tag{3}$$

For instance, on Figure 1b, with $\varepsilon = 2, min_t = 2$ we have two convoys $(\{o_1, o_2\}, \{t_1, t_2, t_3, t_4\})$ and $(\{o_1, o_2, o_3\}, \{t_3, t_4\})$.

Until now, we have considered that we have a group of objects that move close to each other for a long time interval. As shown in [11], moving clusters and different kinds of flocks virtually share essentially the same definition. Basically, the main difference is based on the clustering techniques used. Flocks usually consider a rigid definition of the radius while moving clusters and convoys apply a density-based clustering algorithm (e.g. DBScan [5]). Moving clusters can be seen as special cases of convoys with the additional condition that they need to share some objects between two consecutive timestamps [11]. Therefore, in the following, for brevity and clarity sake we will mainly focus on convoy and density-based clustering algorithms.

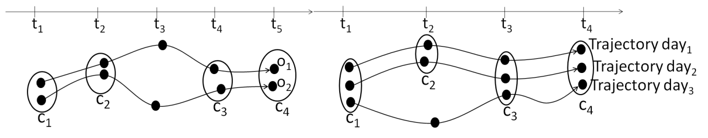
In [14], Hwang et al. propose a general pattern, called a *group pattern*, which essentially is a combination of both convoys and closed swarms. Basically, group pattern is a set of disjointed convoys which are generated by the same group of objects in different time intervals. By considering a convoy as a timepoint, a group pattern can be seen as a swarm of disjointed convoys. Additionally, group pattern cannot be enlarged in terms of objects and number of convoys. Therefore, group pattern is essentially a closed swarm of disjointed convoys. Formally, group pattern can be defined as follows:

Definition 3 *Group Pattern [14]. Given a set of objects O, a minimum weight threshold min_{wei}, a set of disjointed convoys $T_{\mathcal{S}} = \{s_1, s_2, \ldots, s_n\}$, a minimum number of convoys min_c. $(O, T_{\mathcal{S}})$ is a group pattern if:*

$$\begin{cases} (1) : (O, T_{\mathcal{S}}) \textit{ is a closed swarm with } \varepsilon, min_c. \textit{ (e.g. } |T_{\mathcal{S}}| \geq min_c) \\ (2) : \frac{\sum_{i=1}^{|T_{\mathcal{S}}|} |s_i|}{|T_{DB}|} \geq min_{wei}. \end{cases} \tag{4}$$

For instance, see Figure 2a, with $min_t = 2$ and $\varepsilon = 2$ we have a set of convoys $T_{\mathcal{S}} = \{(\{o_1, o_2\}, \{t_1, t_2\}), (\{o_1, o_2\}, \{t_4, t_5\})\}$. Additionally, with $min_c = 1$ we have $(\{o_1, o_2\}, T_{\mathcal{S}})$ is a closed swarm of convoys because $|T_{\mathcal{S}}| = 2 \geq min_c$, $|O| \geq \varepsilon$ and $(O, T_{\mathcal{S}})$ cannot be enlarged. Furthermore, with $min_{wei} = 0.5$, $(O, T_{\mathcal{S}})$ is a group pattern since $\frac{|[t_1,t_2]| + |[t_4,t_5]|}{|T_{DB}|} = \frac{4}{5} \geq min_{wei}$.

Previously, we overviewed patterns in which group objects move together during some time intervals. However, mining patterns from individual object movement is also interesting. In [10], N. Mamoulis et al. propose the notion of *periodic patterns* in which an object follows the same routes (approximately) over regular time intervals. For example, people wake up at the same time and generally follow the same route to



A group pattern example. A periodic pattern example.

Fig. 2. Group pattern and periodic patten example.

their work everyday. Given that an object's trajectory $\mathcal{N}$ is decomposed into $\lfloor \frac{\mathcal{N}}{\mathcal{T_P}} \rfloor$ sub-trajectories. $\mathcal{T_P}$ is data-dependent and has no definite value. For example, $\mathcal{T_P}$ can be set to 'a day' or 'a year' depended on different applications. Essentially, a periodic pattern is a closed swarm discovered from $\lfloor \frac{\mathcal{N}}{\mathcal{T_P}} \rfloor$ sub-trajectories. For instance, in Figure 2b, we have 3 daily sub-trajectories and from them we extract the two following periodic patterns $\{c_1, c_2, c_3, c_4\}$ and $\{c_1, c_3, c_4\}$. As we have provided the definition of a closed swarm, we will mainly focus on closed swarm mining below.

2.2 Related Work

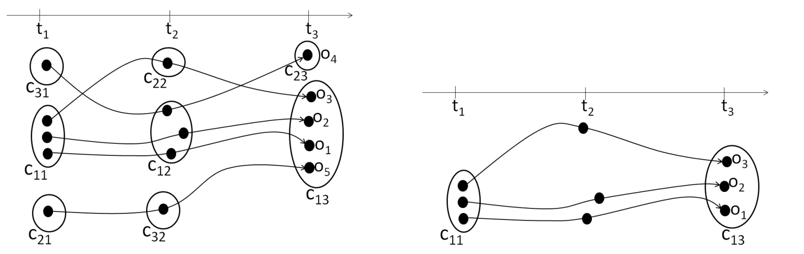
As we mentioned before, many approaches have been proposed to extract patterns. The interested readers may refer to [11] where short descriptions of the most efficient or interesting patterns and approaches are proposed. For instance, Gudmundsson and van Kreveld [1], Vieira et al. [2] define a flock pattern, in which the same set of objects stay together in a circular region with a predefined radius, Kalnis et al. [4] propose the notion of *moving clusters*, while Jeung et al. [3] define a convoy pattern.

Jeung et al. [3] adopt the DBScan algorithm [5] to find candidate convoy patterns. The authors propose three algorithms $CMC, CuTS, CuTS^*$ that incorporate trajectory simplification techniques in the first step. The distance measurements are performed on trajectory segments of as opposed to point based distance measurements. Then, the authors proposed to interpolate the trajectories by creating virtual time points and by applying density measurements on trajectory segments. Additionally, the convoy is defined as a candidate when it has at least *k* clusters during *k* consecutive timestamps.

Recently, Zhenhui Li et al. [6] propose the concept of swarm and closed swarm and the *ObjectGrowth* algorithm to extract closed swarm patterns. The ObjectGrowth method is a depth-first-search of all subsets of O_{DB} through a pre-order tree traversal. To speed up the search process, they propose two pruning rules. *Apriori Pruning* and *Backward Pruning* are used to stop traversal the subtree when we find further traversal that cannot satisfy min_t and closure property. After pruning the invalid candidates, a *ForwardClosure checking* is used to determine whether a pattern is a closed swarm.

In [14], Hwang et al. propose two algorithms to mine group patterns, known as the *Apriori-like Group Pattern mining* algorithm and *Valid Group-Growth* algorithm. The former explores the Apriori property of valid group patterns and extends the Apriori algorithm to mine valid group patterns. The latter is based on idea similar to the FP-growth algorithm. Recently in [7], A. Calmeron proposes a frequent itemset-based approach for flock identification purposes.

Even if these approaches are very efficient they suffer the problem that they only extract a specific kind of pattern. When considering a dataset, it is quite difficult, for



(a) An illustrative example. (b) A swarm from the illustrative example.

Fig. 3. An illustrative example.

Table 2. Cluster Matrix

T_{DB}		t_1			t_2			t_3	
Clusters C_{DB}		c_{11}	c_{21}	c_{31}	c_{12}	c_{22}	c_{32}	c_{13}	c_{23}
O_{DB}	o_1	1			1			1	
	o_2	1			1			1	
	o_3	1				1		1	
	o_4			1	1				1
	o_5		1				1	1	

the decision maker, to know in advance the kind of patterns embedded in the data. Therefore proposing an approach able to automatically extract all these different kinds of patterns can be very useful and this is the problem we address in this paper and that will be developed in the next sections.

3 Spatio-Temporal Patterns in Itemset Context

Basically, patterns are evolution of clusters over time. Therefore, to manage the evolution of clusters, we need to analyse the correlations between them. Furthermore, if clusters share some characteristics (e.g. share some objects), they could be a pattern. Consequently, if a cluster is considered as an item we will have a set of items (called itemset). The main problem essentially is to efficiently combine items (clusters) to find itemsets (a set of clusters) which share some characteristics or satisfy some properties to be considered as a pattern. To describe cluster evolution, spatio-temporal data is presented as a cluster matrix from which patterns can be extracted.

Definition 4 *Cluster Matrix. Assume that we have a set of clusters $C_{DB} = \{C_1, C_2, \ldots, C_n\}$ where $C_i = \{c_{i_1 t_i}, c_{i_2 t_i}, \ldots, c_{i_m t_i}\}$ is a set of clusters at timestamps t_i. A cluster matrix is thus a matrix of size $|O_{DB}| \times |C_{DB}|$. Each row represents an object and each column represents a cluster. The value of the cluster matrix cell, (o_i, c_j) is 1 (resp. empty) if o_i is in (resp. is not in) cluster c_j. A cluster (or item) c_j is a cluster formed after applying clustering techniques.*

For instance, the data from Figure 3a is presented in a cluster matrix in Table 2. Object o_1 belongs to the cluster c_{11} at timestamp t_1. For clarity reasons in the following, c_{ij} represents the cluster c_i at time t_j. Therefore, the matrix cell $(o_1\text{-}c_{11})$ is 1, meanwhile the matrix cell $(o_4\text{-}c_{11})$ is empty because object o_4 does not belong to c_{11}.

By presenting data in a cluster matrix, each object acts as a transaction while each cluster c_j stands for an item. Additionally, an itemset can be formed as $\Upsilon = \{c_{t_{a_1}}, c_{t_{a_2}}, \ldots, c_{t_{a_p}}\}$ with life time $T_\Upsilon = \{t_{a_1}, t_{a_2}, \ldots, t_{a_p}\}$ where $t_{a_1} < t_{a_2} < \ldots < t_{a_p}$, $\forall a_i : t_{a_i} \in T_{DB}, c_{t_{a_i}} \in C_{a_i}$. The support of the itemset Υ, denoted $\sigma(\Upsilon)$, is the number of common objects in every items belonging to Υ, $O(\Upsilon) = \bigcap_{i=1}^{p} c_{t_{a_i}}$. Additionally, the length of Υ, denoted $|\Upsilon|$, is the number of items or timestamps ($= |T_\Upsilon|$). For instance, in Table 2, for a support value of 2 we have: $\Upsilon = \{c_{11}, c_{12}\}$ veryfying $\sigma(\Upsilon) = 2$. Every items (resp. clusters) of Υ, c_{11} and c_{12}, are in the transactions (resp. objects) o_1, o_2. The length of $|\Upsilon|$ is the number of items ($= 2$). Naturally, the number of clusters can be large; however, the maximum length of itemsets is $|T_{DB}|$. Because of the density-based clustering algorithm used, clusters at the same timestamp cannot be in the same itemsets.

Now, we will define some useful properties to extract the patterns presented in Section 2 from frequent itemsets as follows:

Property 1. Swarm. Given a frequent itemset $\Upsilon = \{c_{t_{a_1}}, c_{t_{a_2}}, \ldots, c_{t_{a_p}}\}$. $(O(\Upsilon), T_\Upsilon)$ is a swarm if and only if:

$$\begin{cases} (1) : \sigma(\Upsilon) \geq \varepsilon \\ (2) : |\Upsilon| \geq min_t \end{cases} \tag{5}$$

Proof. After construction, we have $\sigma(\Upsilon) \geq \varepsilon$ and $\sigma(\Upsilon) = |O(\Upsilon)|$ then $|O(\Upsilon)| \geq \varepsilon$. Additionally, as $|\Upsilon| \geq min_t$ and $|\Upsilon| = |T_\Upsilon|$ then $|T_\Upsilon| \geq min_t$. Furthermore, $\forall t_{a_j} \in T_\Upsilon, O(\Upsilon) \subseteq c_{t_{a_j}}$, means that at every timestamp we have a cluster containing all objects in $O(\Upsilon)$. Consequently, $(O(\Upsilon), T_\Upsilon)$ is a swarm because it satisfies all the requirements of the *Definition 1.*

For instance, in Figure 3b, for the frequent itemset $\Upsilon = \{c_{11}, c_{13}\}$ we have $(O(\Upsilon) = \{o_1, o_2, o_3\}, T_\Upsilon = \{t_1, t_3\})$ which is a swarm with support threshold $\varepsilon = 2$ and $min_t = 2$. We can notice that $\sigma(\Upsilon) = 3 > \varepsilon$ and $|\Upsilon| = 2 \geq min_t$.

Property 2. Closed Swarm. Given a frequent itemset $\Upsilon = \{c_{t_{a_1}}, c_{t_{a_2}}, \ldots, c_{t_{a_p}}\}$. $(O(\Upsilon), T_\Upsilon)$ is a closed swarm if and only if:

$$\begin{cases} (1) : (O(\Upsilon), T_\Upsilon) \textit{ is a swarm.} \\ (2) : \nexists \Upsilon' \textit{ s.t } O(\Upsilon) \subset O(\Upsilon'), T_{\Upsilon'} = T_\Upsilon \textit{ and} \\ (O(\Upsilon'), T_\Upsilon) \textit{ is a swarm.} \\ (3) : \nexists \Upsilon' \textit{ s.t. } O(\Upsilon') = O(\Upsilon), T_\Upsilon \subset T_{\Upsilon'} \textit{ and} \\ (O(\Upsilon), T_{\Upsilon'}) \textit{ is a swarm.} \end{cases} \tag{6}$$

Proof. After construction, we obtain $(O(\Upsilon), T_\Upsilon)$ which is a swarm. Additionally, if $\nexists \Upsilon'$ s.t $O(\Upsilon) \subset O(\Upsilon'), T_{\Upsilon'} = T_\Upsilon$ and $(O(\Upsilon'), T_\Upsilon)$ is a swarm then $(O(\Upsilon), T_\Upsilon)$ cannot be enlarged in terms of objects. Therefore, it satisfies the *object-closed* condition. Furthermore, if $\nexists \Upsilon'$ s.t. $O(\Upsilon') = O(\Upsilon), T_\Upsilon \subset T_{\Upsilon'}$ and $(O(\Upsilon), T_{\Upsilon'})$ is a swarm then $(O(\Upsilon), T_\Upsilon)$ cannot be enlarged in terms of lifetime. Therefore, it satisfies the *time-closed* condition. Consequently, $(O(\Upsilon), T_\Upsilon)$ is a swarm and it satisfies *object-closed* and *time-closed* conditions and therefore $(O(\Upsilon), T_\Upsilon)$ is a closed swarm according to the *Definition 1.*

In this paper, we do not provide the properties and proof for convoys, moving clusters which are basically extended by adding some conditions to *Property 1*. For instance, a convoy is a swarm which satisfies the consecutiveness in terms of time condition. For moving clusters [4], they need to share some objects between two timestamps (integrity proportion). Regarding to periodic patterns, the main difference in periodic pattern mining is the input data while the property is similar to *Property 2*. With a slightly modifying cluster matrix such as "each object o becomes a sub-trajectory", we can extract periodic patterns by applying *Property 2*.

Please remember that group pattern is a set of disjointed convoys. Therefore, the group pattern property is as follows:

Property 3. Group Pattern. Given a frequent itemset $\Upsilon = \{c_{t_{a_1}}, c_{t_{a_2}}, \ldots, c_{t_{a_p}}\}$, a mininum weight min_{wei}, a minimum number of convoys min_c, a set of consecutive time segments $T_{\mathcal{S}} = \{s_1, s_2, \ldots, s_n\}$. $(O(\Upsilon), T_{\mathcal{S}})$ is a group pattern if and only if:

$$\begin{cases} (1) : |T_{\mathcal{S}}| \geq min_c. \\ (2) : \forall s_i, s_i \subseteq T_\Upsilon, |s_i| \geq min_t. \\ (3) : \bigcap_{i=1}^{n} s_i = \emptyset, \bigcap_{i=1}^{n} O(s_i) = O(\Upsilon). \\ (4) : \forall s \notin T_{\mathcal{S}}, s \textit{ is a convoy, } O(\Upsilon) \nsubseteq O(s). \\ (5) : \frac{\sum_{i=1}^{n} |s_i|}{|T|} \geq min_{wei}. \end{cases} \tag{7}$$

Proof. If $|T_{\mathcal{S}}| \geq min_c$ then we know that at least min_c consecutive time intervals s_i in $T_{\mathcal{S}}$. Furthermore, if $\forall s_i, s_i \subseteq T_\Upsilon$ then we have $O(\Upsilon) \subseteq O(s_i)$. Additionally, if $|s_i| \geq min_t$ then $(O(\Upsilon), s_i)$ is a convoy (*Definition 2*). Now, $T_{\mathcal{S}}$ actually is a set of convoys of $O(\Upsilon)$ and if $\bigcap_{i=1}^{n} s_i = \emptyset$ then $T_{\mathcal{S}}$ is a set of disjointed convoys. A little bit further, if $\forall s \notin T_{\mathcal{S}}, s$ is a convoy and $O(\Upsilon) \nsubseteq O(s)$ then $\nexists T_{\mathcal{S}'}$ s.t. $T_{\mathcal{S}} \subset T_{\mathcal{S}'}$ and $\bigcap_{i=1}^{|T_{\mathcal{S}'}|} O(s_i) = O(\Upsilon)$. Therefore, $(O(\Upsilon), T_{\mathcal{S}})$ cannot be enlarged in terms of *number of convoys*. Similarly, if $\bigcap_{i=1}^{n} O(s_i) = O(\Upsilon)$ then $(O(\Upsilon), T_{\mathcal{S}})$ cannot be enlarged in terms of *objects*. Consequently, $(O(\Upsilon), T_{\mathcal{S}})$ is a closed swarm of disjointed convoys because $|O(\Upsilon)| \geq \varepsilon, |T_{\mathcal{S}}| \geq min_c$ and $(O(\Upsilon), T_{\mathcal{S}})$ cannot be enlarged (*Definition 1*). Finally, if $(O(\Upsilon), T_{\mathcal{S}})$ satisfies condition (5) then it is a valid group pattern due to *Definition 3*.

Above, we presented some useful properties to extract spatio-temporal patterns from itemsets. Now we will focus on the fact that from an itemset mining algorithm we are able to extract the set of all spatio-temporal patterns. We thus start the proof process by analyzing the swarm extracting problem. This first lemma shows that from a set of frequent itemsets we are able to extract all the swarms embedded in the database.

Lemma 1. *Let $FI = \{\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_l\}$ be the frequent itemsets being mined from the cluster matrix with $minsup = \varepsilon$. All swarms (O, T) can be extracted from FI.*

Proof. Let us assume that (O, T) is a swarm. Note, $T = \{t_{a_1}, t_{a_2}, \ldots, t_{a_m}\}$. According to the *Definition 1* we know that $|O| \geq \varepsilon$. If (O, T) is a swarm then $\forall t_{a_i} \in T, \exists c_{t_{a_i}}$ s.t. $O \subseteq c_{t_{a_i}}$ therefore $\bigcap_{i=1}^{m} c_{t_{a_i}} = O$. Additionally, we know that $\forall c_{t_{a_i}}, c_{t_{a_i}}$ is an item so $\exists \Upsilon = \bigcup_{i=1}^{m} c_{t_{a_i}}$ is an itemset and $O(\Upsilon) = \bigcap_{i=1}^{m} c_{t_{a_i}} = O, T_\Upsilon = \bigcup_{i=1}^{m} t_{a_i} = T$. Therefore, $(O(\Upsilon), T_\Upsilon)$ is a swarm. So, (O, T) is extracted from Υ. Furthermore, $\sigma(\Upsilon) = |O(\Upsilon)| = |O| \geq \varepsilon$ then Υ is a frequent itemset and $\Upsilon \in FI$. Finally, $\forall(O, T)$

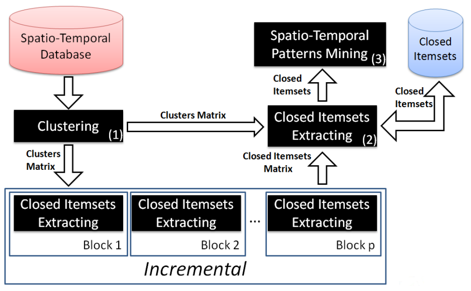


Fig. 4. The main process.

s.t. if (O, T) is a swarm then $\exists \Upsilon$ s.t. $\Upsilon \in FI$ and (O, T) can be extracted from Υ, we can conclude that $\forall$ a swarm (O, T), it can be mined from FI.

We can consider that by adding constraints such as ”consecutive lifetime”, ”time-closed”, ”object-closed”, ”integrity proportion” to swarms, we can retrieve convoys, closed swarms and moving clusters. Therefore, the set of all convoys, closed swarms, moving clusters are subset of the set of all swarms. By applying *Lemma 1*, we retrieve all swarms from frequent itemsets. Consequently, convoys and moving clusters can be completely extracted from frequent itemsets. Additionally, all periodic patterns also can be extracted because they are similar to closed swarms. Furthermore, the following lemma shows that all of group patterns can be extracted from frequent itemsets.

Lemma 2. *Given $FI = \{\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_l\}$ contains all frequent itemsets mined from cluster matrix with $minsup = \varepsilon$. All group patterns $(O, T_{\mathcal{S}})$ can be extracted from FI.*

Proof. $\forall (O, T_{\mathcal{S}})$ is a valid group pattern, we have $\exists T_{\mathcal{S}} = \{s_1, s_2, \ldots, s_n\}$ and $T_{\mathcal{S}}$ is a set of disjointed convoys of O. Therefore, (O, T_{s_i}) is a convoy and $\forall s_i \in T_{\mathcal{S}}, \forall t \in T_{s_i}, \exists c_t$ s.t. $O \subseteq c_t$. Let us assume C_{s_i} is a set of clusters corresponding to s_i, we know that $\exists \Upsilon$, Υ is an itemset, $\Upsilon = \bigcup_{i=1}^{n} C_{s_i}$ and $O(\Upsilon) = \bigcap_{i=1}^{n} O(C_{s_i}) = O$. Additionally, $(O, T_{\mathcal{S}})$ is a valid group pattern; therefore, $|O| \geq \varepsilon$ so $|O(\Upsilon)| \geq \varepsilon$. Consequently, Υ is a frequent itemset and $\Upsilon \in FI$ because Υ is an itemset and $\sigma(\Upsilon) = |O(\Upsilon)| \geq \varepsilon$. Consequently, $\forall (O, T_{\mathcal{S}}), \exists \Upsilon \in FI$ s.t. $(O, T_{\mathcal{S}})$ can be extracted from Υ and therefore all group patterns can be extracted from FI.

4 FCI-based Spatio-Temporal Pattern Mining Algorithm

In this section, we propose two approaches i.e., *GeT_Move* and *Incremental GeT_Move*, to efficiently extract patterns. The global process is illustrated in Figure 4. In the first step, a clustering approach is applied at each timestamp to group objects into different clusters. For each timestamp t_a, we thus have a set of clusters $C_a = \{c_{1t_a}, c_{2t_a}, \ldots, c_{mt_a}\}$, with $1 \leq k \leq m, c_{kt_a} \subseteq O_{DB}$. Spatio-temporal data can thus be converted to a cluster matrix CM.

4.1 GeT_Move

After generating the cluster matrix CM, a FCI mining algorithm is applied on CM to extract all the FCIs. By scanning them and checking properties, we can obtain the

Algorithm 1: GeT_Move

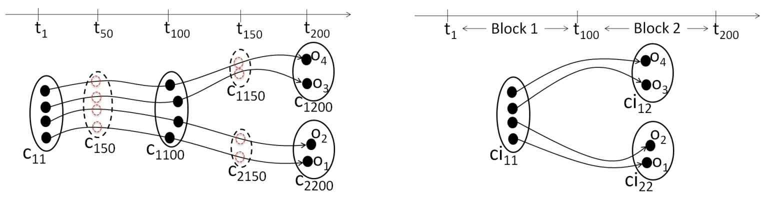
```
Input : int $\varepsilon$, int $min_t$, set of items $C_{DB}$, double $\theta$, int $min_c$, double $min_{wei}$
begin
    LCM_PatternMining($C_{DB}$, $\varepsilon$);
PatternMining($X$, $min_t$)
begin
    if $|X| \geq min_t$ then
        output $X$; /*Closed Swarm*/
        $gPattern := \emptyset$; $convoy := \emptyset$; $mc := \emptyset$;
        for $k := 1$ to $|X - 1|$ do
            if $x_k.time = x_{(k+1)}.time - 1$ then
                $convoy := convoy \cup x_k$;
                if $\frac{|\mathcal{T}(x_k) \cap \mathcal{T}(x_{k+1})|}{|\mathcal{T}(x_k) \cup \mathcal{T}(x_{k+1})|} \geq \theta$ then
                    $mc := mc \cup x_k$;
                else
                    if $|mc \cup x_k| \geq min_t$ then
                        output $mc \cup x_k$; /*MovingCluster*/
                    $mc := \emptyset$;
            else
                if $|convoy \cup x_k| \geq min_t$ and $|\mathcal{T}(convoy \cup x_k)| = |\mathcal{T}(X)|$ then
                    output $convoy \cup x_k$; /*Convoy*/
                    $gPattern := gPattern \cup (convoy \cup x_k)$;
                if $|mc \cup x_k| \geq min_t$ then
                    output $mc \cup x_k$; /*MovingCluster*/
                $convoy := \emptyset$; $mc := \emptyset$;
        if $|gPattern| \geq min_c$ and $size(gPattern) \geq min_{wei}$ then
            output $gPattern$; /*Group Pattern*/
Where: $X$ is itemset, $\mathcal{T}(X)$ is list of tractions that $X$ belongs to, $x_k.time$ is time index of item $x_k$,
$|\mathcal{T}(convoy)|$ is the number of transactions that the $convoy$ belongs to, $|gPattern|$ and $size(gPattern)$
respectively are the number of convoys and the proportion of total length of the convoys in $gPattern$ to $T_{DB}$.
```

Algorithm 2: Incremental GeT_Move

```
Input : int $\varepsilon$, int $min_t$, double $\theta$, set of Occurrence sets (blocks) $B$, int $min_c$, double $min_{wei}$
begin
    $K := \emptyset$; $CI := \emptyset$;
    foreach $b \in B$ do
        CI := CI.update(LCM($b$, $\varepsilon$));
    GeT_Move($\varepsilon$, $min_t$, $CI$, $\theta$, $min_c$, $min_{wei}$);
```

patterns. In this paper, we apply the LCM algorithm [8] to extract FCIs as it is known to be a very efficient algorithm. In LCM algorithm's process, we discard some useless candidate itemsets. In spatio-temporal patterns, items (resp. clusters) must belong to different timestamps and therefore items (resp. clusters) which form a FCI must be in different timestamps. In contrast, we are not able to extract patterns by combining items in the same timestamp. Consequently, FCIs which include more than 1 item in the same timestamp will be discarded.

Thanks to the above characteristic, we now have the maximum length of the FCIs which is the number of timestamps $|T_{DB}|$. Additionally, the LCM search space only depends on the number of objects (transactions) $|O_{DB}|$ and the maximum length of itemsets $|T_{DB}|$. Consequently, by using LCM and by applying the above characteristic, *GeT_Move* is not affected by the number of clusters and therefore the computing time can be greatly reduced.



(a) The entire dataset (b) Data after applying FCI mining on blocks

Fig. 5. A case study example. (b)-$ci_{11}, ci_{12}, ci_{22}$ are FCIs extracted from block 1 and block 2.

Table 3. Closed Itemset Matrix

Block B		b_1	b_2	
Frequent Closed Itemsets CI		ci_{11}	ci_{12}	ci_{22}
O_{DB}	o_1	1		1
	o_2	1		1
	o_3	1	1	
	o_4	1	1	

The pseudo code of *GeT_Move* is described in *Algorithm 1*. The core of *GeT_Move* algorithm is based on the LCM algorithm which has been modified by adding the pruning rule and by extracting patterns from FCIs (line 2). Whenever a closed itemset is detected, the PatternMining sub-function (lines 3-25) is invoked to check properties of the itemset X to extract spatio-temporal patterns.

4.2 Incremental GeT_Move

Naturally, in real world applications (cars, animal migration), the objects tend to move together in short interval meanwhile their movements can be different in long interval. Therefore, the number of items (clusters) can be large and the length of FCIs can be long. Additionally, refer to [13, 8], the FCI mining algorithms search space are affected by the number of items and the length of itemsets. For instance, see Figure 5a, objects $\{o_1, o_2, o_3, o_4\}$ move together during first 100 timestamps and after that o_1, o_2 stay together while o_3, o_4 move together in another direction. The problem here is that if we apply *GeT_Move* on the whole dataset, the extraction of the itemsets can be very time consuming.

To deal with the issue, we propose the *Incremental GeT_Move* algorithm. The main idea is to split the trajectories (resp. cluster matrix CM) into short intervals, called blocks. By applying FCI mining on each short interval, the data can then be compressed into local FCIs. Additionally, the length of itemsets and the number of items can be greatly reduced. For instance, see Figure 5, if we consider $[t_1, t_{100}]$ as a block and $[t_{101}, t_{200}]$ as another block, the maximum length of itemsets in both blocks is 100 (instead of 200). Additionally, the original data can be greatly compressed (e.g. Figure 5b) and only 3 items remain: $ci_{11}, ci_{12}, ci_{22}$.

Definition 5 *Block. Given a set of timestamps $T_{DB} = \{t_1, t_2, \ldots, t_n\}$, a cluster matrix CM. CM is vertically split into equivalent (in terms of intervals) smaller cluster matrices and each of them is a block b. Assume T_b is a set of timestamps of block b, $T_b = \{t_1, t_2, \ldots, t_k\}$, thus we have $|T_b| = k \leq |T_{DB}|$.*

Assume that we obtain a set of blocks $B = \{b_1, b_2, \dots, b_p\}$ with $|T_{b_1}| = |T_{b_2}| = \dots = |T_{b_p}|, \bigcup_{i=1}^{p} b_i = CM$ and $\bigcap_{i=1}^{p} b_i = \emptyset$. Given a set of FCI collections $CI = \{CI_1, CI_2, \dots, CI_p\}$ where CI_i is mined from block b_i. CI is presented as a *closed itemset matrix* which is formed by horizontally connecting all local FCIs: $CIM = \bigcup_{i=1}^{p} CI_i$.

Definition 6 *Closed Itemset Matrix (CIM). Closed itemset matrix is a cluster matrix with some differences as follows: 1) Timestamp t now becomes a block b. 2) Item c is a FCI ci.*

For instance, see Table 3, we have two sets of FCIs $CI_1 = \{ci_{11}\}, CI_2 = \{ci_{12}, ci_{22}\}$ which are respectively extracted from blocks b_1, b_2. We have CIM which is created from CI_1, CI_2 in Table 3.

Now, by applying FCI mining on closed itemset matrix CIM, we retrieve all FCIs from corresponding data. Note that items (in CIM) which are in the same block cannot be in the same FCIs.

Lemma 3. *Given a cluster matrix CM which is vertically split into a set of blocks $B = \{b_1, b_2, \dots, b_p\}$ so that $\forall \Upsilon, \Upsilon$ is a FCI and Υ is extracted from CM then Υ can be extracted from the closed itemset matrix CIM.*

Proof. Let us assume that $\forall b_i, \exists I_i$ is a set of items belonging to b_i and therefore we have $\bigcap_{i=1}^{|B|} I_i = \emptyset$. If $\forall \Upsilon, \Upsilon$ is a FCI extracted from CM then Υ is formed as $\Upsilon = \{\gamma_1, \gamma_2, \dots, \gamma_p\}$ where γ_i is a set of items s.t. $\gamma_i \subseteq I_i$. Additionally, Υ is a FCI and $O(\Upsilon) = \bigcap_{i=1}^{p} O(\gamma_i)$ then $\forall O(\gamma_i), O(\Upsilon) \subseteq O(\gamma_i)$. Furthermore, we have $|O(\Upsilon)| \geq \varepsilon$; therefore, $|O(\gamma_i)| \geq \varepsilon$ so γ_i is a frequent itemset. Assume that $\exists \gamma_i, \gamma_i \notin CI_i$ then $\exists \Psi, \Psi \in CI_i$ s.t. $\gamma_i \subseteq \Psi$ and $\sigma(\gamma_i) = \sigma(\Psi), O(\gamma_i) = O(\Psi)$. Note that Ψ, γ_i are from b_i. Remember that $O(\Upsilon) = O(\gamma_1) \cap O(\gamma_2) \cap \dots \cap O(\gamma_i) \cap \dots \cap O(\gamma_p)$ and we have: $\exists \Upsilon'$ s.t. $O(\Upsilon') = O(\gamma_1) \cap O(\gamma_2) \cap \dots \cap O(\Psi) \cap \dots \cap O(\gamma_p)$. Therefore, $O(\Upsilon') = O(\Upsilon)$ and $\sigma(\Upsilon') = \sigma(\Upsilon)$. Additionally, we know that $\gamma_i \subseteq \Psi$ so $\Upsilon \subseteq \Upsilon'$. Consequently, we obtain $\Upsilon \subseteq \Upsilon'$ and $\sigma(\Upsilon) = \sigma(\Upsilon')$. Therefore, Υ is not a FCI. That violates the assumption and therefore we have: if $\exists \gamma_i, \gamma_i \notin CI_i$ therefore Υ is not a FCI. Finally, we can conclude that $\forall \Upsilon, \Upsilon = \{\gamma_1, \gamma_2, \dots, \gamma_p\}$ is a FCI extracted from CM, $\forall \gamma_i \in \Upsilon$, γ_i must be belong to CI_i and γ_i is an item in closed itemset matrix CIM. Therefore, Υ can be retrieved by applying FCI mining on CIM.

By applying *Lemma 3*, we can obtain all the FCIs and from the itemsets, patterns can be extracted. Note that the Incremental GeT_Move does not depend on the length restriction min_t. The reason is that min_t is only used in Spatio-Temporal Patterns Mining step. Whatever min_t ($min_t \geq$ block size or $min_t \leq$ block size), Incremental GeT_Move can extract all the FCIs and therefore the final results are the same.

The pseudo code of *Incremental GeT_Move* is described in *Algorithm 2*. The main different between the code of *Incremental GeT_Move* and *GeT_Move* is the *update* function. In this function, we step by step generate the closed itemsets matrix from extracted closed itemsets in blocks (line 4). Next, we apply *GeT_Move* to extract patterns (line 5).

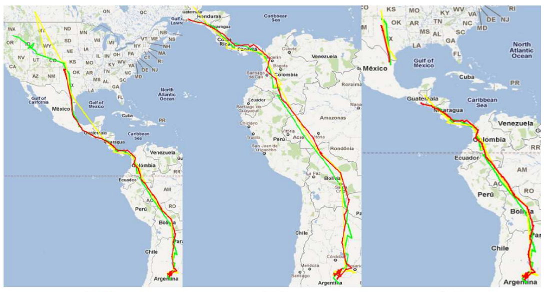


Fig. 6. An example of patterns discovered from Swainsoni dataset. (a) One of discovered closed swarms, (b) One of discovered convoys, (c) One of discovered group patterns.

5 Experimental Results

A comprehensive performance study has been conducted on real datasets 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 (source code) is available and also integrated in our online demonstration system[4]. As in [6], we only report the results on the following datasets[5]: *Swainsoni dataset* includes 43 objects evolving over time and 764 different timestamps. The interested readers may refer to our online demonstration system[2] for other experimental results. Additionally, similar to [6,3], we first use linear interpolation to fill in the missing data and then DBScan [5] ($MinPts = 2, Eps = 0.001$) is applied to generate clusters at each timestamp.

In the comparison, we employ $CMC, CuTS^*$[6][3] (convoy mining) and $Object$ $Growth$ (closed swarm mining). Note that, in [6], $ObjectGrowth$ outperforms $VG-Growth$ [14] (a group patterns mining algorithm) in terms of performance and therefore we will only consider $ObjectGrowth$ and not both.

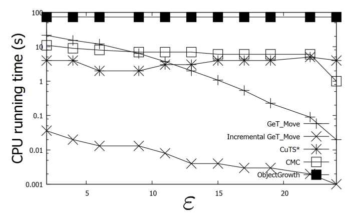
5.1 Effectiveness

We proved that mining spatio-temporal patterns can be similarly mapped into itemsets mining issue. Therefore, in theoretical way, our approaches can provide the correct results. Experimentally, we do a further comparison, we first obtain the spatio-temporal patterns by applying $CMC, CuTS^*$, $ObjectGrowth$ as well as our approaches. To apply our algorithms, we split cluster matrix into blocks such as each block b contains 25 timestamps. Additionally, to retrieve all the spatio-temporal patterns, in the reported experiments, the default value of ε is set to 2 (two objects can form a pattern), min_t is 1. Note that the default values are the hardest conditions for examining the algorithms. Then in the following we mainly focus on different values of min_t in order to obtain different sets of convoys, closed swarms and group patterns. Note that for group patterns, min_c is 1 and min_{wei} is 0.

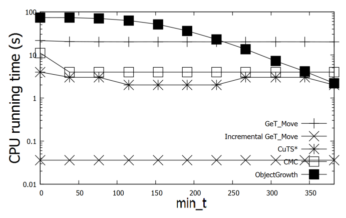
[4] www.lirmm.fr/∼phan/index.jsp

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

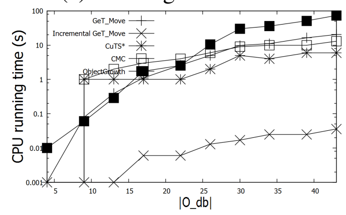
[6] The source code of $CMC, CuTS^*$ is available at http://lsirpeople.epfl.ch/jeung/source_codes.htm



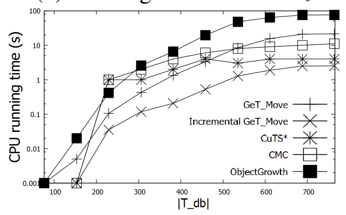
(a) Running time w.r.t. ε



(b) Running time w.r.t. min_t



(c) Running time w.r.t. $|O_{DB}|$



(d) Running time w.r.t. $|T_{DB}|$

Fig. 7. Running time on Swainsoni Dataset.

The results show that our proposed approaches obtain the same results compared to the traditional algorithms. An example of patterns is illustrated in Figure 6. For instance, see Figure 6a, a closed swarm is discovered within a FCI. Furthermore, from the itemset, a convoy and a group pattern are also extracted (i.e. Figure 6b, 6c).

5.2 Efficiency

In the reported experiments, GeT_Move and Incremental GeT_Move extract closed swarms, convoys and group patterns while $CMC, CuTS^*$ only extract convoys and $ObjectGrowth$ only extracts closed swarms. Additionally, all the algorithms are applied on cluster matrices, thus clustering cost was not taken into account.

Efficiency w.r.t. ε, min_t. Figure 7a shows running time w.r.t. ε. It is clear that our approaches outperform other algorithms. ObjectGrowth is the lowest one and the main reason is that with low min_t (default $min_t = 1$), the *Apriori Pruning* rule (the most efficient pruning rule) is no longer effective. Therefore, the search space is greatly enlarged ($2^{|O_{DB}|}$ in the worst case). Additionally, there is no pruning rule for ε and therefore the change of ε does not directly affect the running time of ObjectGrowth. A little bit further, GeT_Move is lower than Incremental GeT_Move. The main reason is that GeT_Move has to process with large number of items and long itemsets. While, thanks to blocks, the number of items is greatly reduced and itemsets are not long as the ones in GeT_Move. Figure 7b shows running time w.r.t. min_t. In almost all cases, our approaches outperform other algorithms.

Efficiency w.r.t. $|O_{DB}|, |T_{DB}|$. Figure 7c-d show the running time when varying $|O_{DB}|$ and $|T_{DB}|$ respectively. In all figures, Incremental GeT_Move outperforms other algorithms.

In addition, the other experimental results on different real and synthetic datasets are available on our demo system website. Some interesting experiments on the number of patterns, algorithm scalability, optimal block-sizes and also a parameter free version of Incremental GeT_Move are integrated on the online system. Interested readers may refer to "www.lirmm.fr/ $\sim$phan/GeTMove.html".

6 Conclusion and Discussion

In this paper, we propose unifying incremental approaches to automatically extract different kinds of spatio-temporal patterns by applying FCI mining techniques. Their effectiveness and efficiency have been evaluated by using real and synthetic datasets. Experiments show that our approaches outperform traditional ones.

One next issue we plan to address is how to take into account the arrival of new objects which were not available for the first extraction. Now, we can store the result to improve the process when new object movements arrive. But, in this approach, we take the hypothesis is that the number of objects remains the same. However in some applications these objects could be different.

References

1. J. Gudmundsson, M.van Kreveld. *Computing longest duration flocks in trajectory data*. In: GIS 06, New York, NY, USA, pp.35-42.
2. MR. Vieira, P. Bakalov, VJ. Tsotras. *On-line Discovery of Flock Patterns in Spatio-Temporal Data*. In: GIS 09, New York, USA, pp.286-295.
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*. KDD '96, Portland, pp. 226-231.
6. Z. Li, B. Ding, J. Han, R. Kays. *Swarm: Mining Relaxed Temporal Moving Object Clusters*. VLDB2010, Singapore, pp. 723-734.
7. A.O.C. Romero. *Mining moving flock patterns in large spatio-temporal datasets using a frequent pattern mining approach*. Master Thesis, University of Twente, March 2011.
8. T. Uno, M. Kiyomi, and H. Arimura. *LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets*. ICDM FIMI 2004.
9. Z. Li, M. Ji, J.-G. Lee, L. Tang, Y. Yu, J. Han, and R. Kays. *Movemine: Mining moving object databases*. In SIGMOD 2010, Indianapolis, Indiana, pp.1203-1206.
10. N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, D. W. Cheung. *Mining, Indexing, and Querying Historical Spatiotemporal Data*. SIGKDD'04, pp.236-245.
11. J. Han, Z. Li, L. A. Tang. *Mining Moving Object, Trajectory and Traffic Data*. In DASFAA'10, Japan.
12. C.S. Jensen, D. Lin, and B.C. Ooi. *Continuous clustering of moving objects*. In KDE(2007), pp. 1161-1174. issn: 1041-4347.
13. C. Lucchese, S. Orlando, R. Perego. *DCI Closed: A Fast and Memory Efficient Algorithm to Mine Frequent Closed Itemsets*. ICDM FIMI 2004.
14. Y. Wang, E.-P. Lim, and S.-Y. Hwang. *Efficient Mining of Group Patterns from User Movement Data*. In DKE(2006), pp. 240-282.