Mining Representative Movement Patterns through Compression

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

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

Abstract. Mining trajectories (or moving object patterns) from spatio-temporal data is an active research field. Most of the researches are devoted to extract trajectories that differ in their structure and characteristic in order to capture different object behaviors. The first issue is constituted from the fact that all these methods extract thousand of patterns resulting in a huge amount of redundant knowledge that poses limit in their usefulness. The second issue is supplied from the nature of spatio-temporal database from which different types of patterns could be extracted. This means that using only a single type of patterns is not sufficient to supply an insightful picture of the whole database.

Motivating by these issues, we develop a Minimum Description Length (MDL)-based approach that is able to compress spatio-temporal data combining different kinds of moving object patterns. The proposed method results in a rank of the patterns involved in the summarization of the dataset. In order to validate the quality of our approach, we conduct an empirical study on real data to compare the proposed algorithms in terms of effectiveness, running time and compressibility.

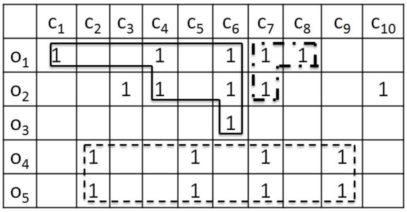
Keywords: MDL, moving objects, spatio-temporal data, top-k, compressibility.

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 [5] [10] [6] is to analyze such datasets for interesting moving object clusters. A moving object cluster can be defined as a group of moving objects that are physically closed to each other for at least some number of timestamps. In this context, many recent studies have been defined such as flocks [5], convoy queries [7], closed swarms [10], group patterns [15], gradual trajectory patterns [6], traveling companions [13], gathering patterns [16], etc...

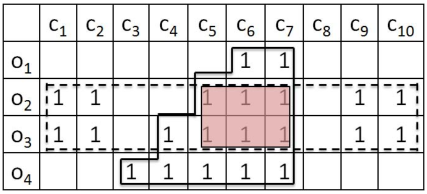
Nevertheless, after the extraction, the end user can be overwhelmed by a huge number of movement patterns although only a few of them are useful. However, relatively few researchers have addressed the problem of reducing movement pattern redundancy. In another context, i.e. frequent itemsets, the Krimp algorithm [14], using the minimum description length (MDL) principle [4], proposes to reduce the amount of itemsets by

J. Pei et al. (Eds.): PAKDD 2013, Part I, LNAI 7818, pp. 314–326, 2013.



	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
o_1	1			1		1	1	1		
o_2			1	1		1	1			1
o_3						1				
o_4		1			1		1		1	
o_5		1			1		1		1	

Fig. 1. An example of moving object database. Shapes are movement patterns, o_i, c_i respectively are objects and clusters.



	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
o_1						1	1			
o_2	1	1			1	1	1		1	1
o_3	1	1		1	1	1	1		1	1
o_4			1	1	1	1	1			

Fig. 2. An example of pattern overlapping, between closed swarm (dashed line rectangle) and $rGpattern^{\geq}$ (step shape), overlapping clusters are c_5, c_6 and c_7

using an efficient encoding and then provide the end-user only with a set of informative patterns.

In this paper, we adapt the MDL principle for mining representative movement patterns. However, one of the key challenges in designing an MDL-based algorithm for moving object data is that the encoding scheme needs to deal with different pattern structures which can cover different parts of the data. If we only consider different kinds of patterns individually then it is difficult to obtain an optimal set of compression patterns.

For instance, see Figure 1, we can notice that there are three different patterns, with different structures, that cover different parts of the moving object data. If we only keep patterns having a rectangular shape then we lose the other two patterns and viceversa.

Furthermore, although patterns express different kinds of knowledge, they can overlap each other as well. Thus, enforcing non-overlapping patterns may result in losing interesting patterns. For instance, see Figure 2, there are two overlapping patterns. Krimp algorithm does not allow overlapping patterns then it has to select one and obviously loses the other one. However, they express very different knowledge and thus, by removing some of them, we cannot fully understand the object movement behavior. Therefore, the proposed encoding scheme must to appropriately deal with the pattern overlapping issue.

Motivated by these challenges, we propose an overlapping allowed multi-pattern structure encoding scheme which is able to compress the data with different kinds of patterns. Additionally, the encoding scheme also allows overlapping between different kinds of patterns. To extract compression patterns, a naive greedy approach, named NAIVECOMPO, is proposed. To speed up the process, we also propose the SMARTCOMPO algorithm which takes into account several useful properties to avoid useless computation. Experimental results on real-life datasets demonstrate the effectiveness and efficiency of the proposed approaches by comparing different sets of patterns.

2 Preliminaries and Problem Statement

2.1 Object Movement Patterns

Object movement patterns are designed to group similar trajectories or objects which tend to move together during a time interval. In the following, we briefly present the definitions of different kinds of movement patterns.

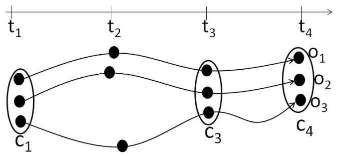


Fig. 3. An example of closed swarm

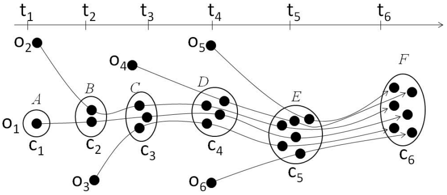


Fig. 4. An example of rGpattern

Database of clusters. Let us consider a set objects occurring at different timestamps. 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 $\{t_1, t_2, \ldots, t_m\}$. Given a cluster $c \in C_{t'}(\in C_{DB})$, $t(c)$ and $o(c)$ are respectively used to denote the timestamp that c is involved in and the set of objects included in c. For brevity sake, we take clustering as a preprocessing step.

After generating C_{DB}, the moving object database (O_{DB}, T_{DB}) is defined such as each object $o \in O_{DB}$ contains a list of clusters (i.e. $o = c_1 c_2 \ldots c_m$) and T_{DB} stands for the associated timestamp. For instance, Figure 1 presents the database O_{DB} and object o_1 can be represented as $o_1 = c_1 c_4 c_6 c_7 c_8$.

From this set different patterns can be extracted. In an informal way, a closed swarm is a list of clusters $cs = c_1 \ldots c_n$ such that they share at least ε common objects, cs contains at least min_t clusters and cs cannot be enlarged in terms of objects and clusters. Note that there are no pairs of clusters which are in the same timestamps involved in cs. Then a closed swarm can be formally defined as follows:

Definition 1. *ClosedSwarm[10]. A list of clusters $cs = c_1 \ldots c_n$ is a closed swarm if:*

$$\begin{cases} (1) : |O(cs)| = |\bigcap_{i=1}^{n} c_i| \geq \varepsilon. \\ (2) : |cs| \geq min_t. \\ (3) : \nexists i, j \in \{1, \ldots, n\}, i \neq j, t(c_i) = t(c_j). \\ (4) : \nexists cs' : cs \subset cs', cs' \textit{ satisfies the conditions (1), (2) and (3).} \end{cases} \tag{1}$$

For instance, see Figure 3, $cs = c_1 c_3 c_4$ is a closed swarm with $min_t = 2, \varepsilon = 2$. Similarly, in Figure 1, we also have $cs = c_2 c_5 c_7 c_9$ is a closed swarm. A convoy is a group of objects such that these objects are closed each other during at least min_t *consecutive* time points. Another pattern is group pattern which essentially is a set of disjointed convoys which are generated by the same group of objects in different time intervals. In this paper, we only consider closed swarm instead of convoy and group pattern since closed swarm is more general [10].

A gradual trajectory pattern [6], denoted *rGpattern*, is designed to capture the gradual object moving trend. More precisely, a rGpattern is a maximal list of moving object clusters which satisfy 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 and the integrity condition can be that all the objects should remain in the next cluster. A rGpattern can be defined as follows:

Definition 2. *rGpattern [6]. Given a list of clusters $C^* = c_1 \ldots c_n$. C^* is a gradual trajectory pattern if:*

$$C^* = C^{\geq} \begin{cases} (1) : |C^*| \geq min_t. \\ \forall i \in \{1, \ldots, n-1\}, \\ (2) : o(c_i) \subseteq o(c_{i+1}). \\ (3) : |c_n| > |c_1|. \\ (4) : \nexists c_m : C^* \cup c_m \; is \; a \; C^{\geq}. \end{cases} \qquad C^* = C^{\leq} \begin{cases} (1) : |C^*| \geq min_t. \\ \forall i \in \{1, \ldots, n-1\}, \\ (2) : o(c_i) \supseteq o(c_{i+1}). \\ (3) : |c_n| < |c_1|. \\ (4) : \nexists c_m : C^* \cup c_m \; is \; a \; C^{\geq}. \end{cases}$$

Essentially, we have two kinds of rGpatterns, $rGpattern^{\geq}$ and $rGpattern^{\leq}$. For instance, see Figure 1, $rGpattern^{\geq} = c_1c_4c_6$ and $rGpattern^{\leq} = c_7c_8$.

2.2 Problem Statement

Eliminating the number of uninteresting patterns is an emerging task in many real world cases. One of the proposed solutions is the MDL principle [4]. Let us start explaining this principle in the following definition:

Definition 3. *(Hypothesis). A hypothesis $\mathcal{P}$ is a set of patterns $\mathcal{P} = \{p_1, p_2, \ldots, p_h\}$.*

Given a scheme S, let $L_S(P)$ be the description length of hypothesis $\mathcal{P}$ and $L_S(O_{DB}|P)$ be the description length of data O_{DB} when encoded with the help of the hypothesis and an encoding scheme S. Informally, the MDL principle proposes that the best hypothesis always compresses the data most. Therefore, the principle suggests that we should look for hypothesis $\mathcal{P}$ and the encoding scheme S such that $L_S(O_{DB}) = L_S(\mathcal{P}) + L_S(O_{DB}|\mathcal{P})$ is minimized. For clarity sake, we will omit S when the encoding scheme is clear from the context. Additionally, the description length of O_{DB} given $\mathcal{P}$ is denoted as $L_{\mathcal{P}}(O_{DB}) = L(\mathcal{P}) + L(O_{DB}|\mathcal{P})$.

In this paper, the hypothesis is considered as a dictionary of movement patterns $\mathcal{P}$. Furthermore, as in [9], we assume that any number or character in data has a fixed length bit representation which requires a unit memory cell. In our context, the description length of a dictionary $\mathcal{P}$ can be calculated as the total lengths of the patterns and the number of patterns (i.e. $L(\mathcal{P}) = \sum_{p \in \mathcal{P}} |p| + |\mathcal{P}|$). Furthermore, the length of the data O_{DB} when encoded with the help of dictionary $\mathcal{P}$ can be calculated as $L(O_{DB}|\mathcal{P}) = \sum_{o \in O_{DB}} |o|$.

The problem of finding compressing patterns can be formulated as follows:

Definition 4. *(Compressing Pattern Problem). Given a moving object database O_{DB}, a set of pattern candidates $F = \{p_1, p_2, \ldots, p_m\}$. Discover an optimal dictionary $\mathcal{P}^*$ which contains at most K movement patterns so that:*

$$\mathcal{P}^* = \arg\min_{\mathcal{P}} \big(L^*_{\mathcal{P}}(O_{DB})\big) = \arg\min_{\mathcal{P}} \big(L^*(\mathcal{P}) + L^*(O_{DB}|\mathcal{P})\big), \mathcal{P}^* \subseteq F \qquad (2)$$

A key issue in designing an MDL-based algorithm is: how can we encode data given a dictionary? The fact is that if we consider closed swarms individually, Krimp algorithm can be easily adapted to extract compression patterns. However, the issue here is that we have different patterns (i.e. closed swarms and rGpatterns) and Krimp algorithm has not been designed to deal with rGpatterns. It does not supply multi-pattern types in the dictionary that may lead to losing interesting ones. Furthermore, as mentioned before, we also have to address the pattern overlapping issue. In this work, we propose a novel overlapping allowed multi-pattern structures encoding scheme for moving object data.

3 Encoding Scheme

3.1 Movement Pattern Dictionary-Based Encoding

Before discussing our encoding for moving object data, we revisit the encoding scheme used in the Krimp algorithm [14]. An itemset I is encoded with the help of itemset patterns by replacing every non-overlapping instance of a pattern occurring in I with a pointer to the pattern in a code table (dictionary). In this way, an itemset can be encoded to a more compact representation and decoded back to the original itemset.

Table 1. An illustrative example of database and dictionary in Figure 1. $\bar{0}$, $\bar{1}$ and $\bar{2}$ respectively are pattern types: closed swarm, $rGpattern^{\geq}$ and $rGpattern^{\leq}$.

O_{DB}	Encoded O_{DB}	Dictionary $\mathcal{P}$
$o_1 = c_1c_4c_6c_7c_8$	$o_1 = [p_1, 0][p_3, 1]$	
$o_2 = c_3c_4c_6c_7c_{10}$	$o_2 = c_3[p_1, 1][p_3, 0]c_{10}$	$p_1 = c_1c_4c_6, \bar{1}$
$o_3 = c_6$	$o_3 = [p_1, 2]$	$p_2 = c_2c_5c_7c_9, \bar{0}$
$o_4 = c_2c_5c_7c_9$	$o_4 = p_2$	$p_3 = c_7c_8, \bar{2}$
$o_5 = c_2c_5c_7c_9$	$o_5 = p_2$	

In this paper we use a similar dictionary-based encoding scheme for moving object database. Given a dictionary consisting of movement patterns $\mathcal{P} = \{p_1, \ldots, p_m\}$, an object $o \in O_{DB}$ containing a list of clusters is encoded by replacing instances of any pattern p_i in o with pointers to the dictionary. An important difference between itemset data and moving object data is that there are different kinds of movement patterns which have their own characteristic. The fact is that if a closed swarm cs occurs in an object o then all the clusters in cs are involved in o. While an object can involve in only a part of a rGpattern and viceversa.

For instance, see Figure 1, we can consider that o_2 joins the $rGpattern^{\geq} = c_1c_4c_6$ at c_4c_6. While, the closed swarm $cs = c_2c_5c_7c_9$ occurs in o_4 and o_5 entirely.

Property 1. (Encoding Properties). Given an object o which contains a list of clusters and a pattern $p = c_1 \ldots c_n$. p occurs in o or o contributes to p if:

$$\begin{cases} (1) : p \textit{ is a } rGpattern^{\geq}, \exists i \in [1, n] \big| \forall j \geq i, c_j \in o. \\ (2) : p \textit{ is a } rGpattern^{\leq}, \exists i \in [1, n] \big| \forall j \leq i, c_j \in o. \\ (3) : p \textit{ is a closed swarm}, \forall j \in [1, n], c_j \in o. \end{cases} \quad (3)$$

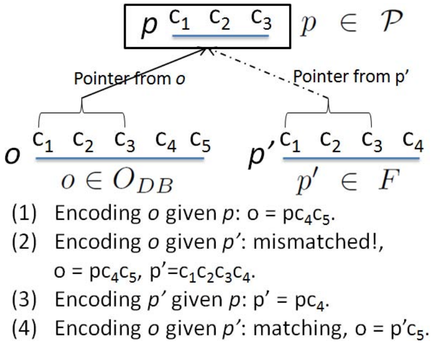
Proof. Case (1): after construction we have $o(c_i) \subseteq o(c_{i+1}) \subseteq \ldots \subseteq o(c_n)$. Additionally, $o \in o(c_i)$. Consequently, $o \in o(c_{i+1}), \ldots, o(c_n)$ and therefore $\forall j \geq i, c_j \in o$. Furthermore, in Case (2): we have $o(c_1) \supseteq o(c_2) \supseteq \ldots \supseteq o(c_{i-1})$. Additionally, $o \in o(c_{i-1})$. Consequently, $o \in o(c_1), \ldots, o(c_{i-1})$ and therefore $\forall j \leq i, c_j \in o$. In Case (3), we have $o \in O(cs) = \bigcap_{i=1}^{n} c_i$ and therefore $\forall j \in [1, n], c_j \in o$.

For instance, see Table 1, we can see that for each pattern, we need to store an extra bit to indicate the pattern type. Regarding to closed swarm, by applying Property 1, in the object o we only need to replace all the clusters, which are included in closed swarm, by a pointer to the closed swarm in the dictionary. However, in gradual trajectories (i.e. $rGpattern^{\geq}$, $rGpattern^{\leq}$), we need to store with the pointer an additional index to indicate the cluster c_i. Essentially, c_i plays the role of a starting involving point (resp. ending involving point) of the object o in a $rGpattern^{\geq}$ (resp. $rGpattern^{\leq}$).

As an example, consider dictionary $\mathcal{P}$ in Table 1. Using $\mathcal{P}$, o_1 can be encoded as $o_1 = [p_1, 0][p_3, 1]$ where 0 (in $[p_1, 0]$) indicates the cluster at index 0 in p_1, (i.e. c_1) and 1 (in $[p_3, 1]$) indicates the cluster at index 1 in p_3, i.e. c_8. While, o_4 can be encoded as $o_4 = p_2$, i.e. p_2 is a closed swarm.

3.2 Overlapping Movement Pattern Encoding

Until now, we have already presented the encoding function for different patterns when encoding an object o given a pattern p. In this section, the encoding scheme will be completed by addressing the pattern overlapping problem so that overlapped patterns can exist in the dictionary $\mathcal{P}$.



(1) Encoding o given p: o = pc$_4$c$_5$.
(2) Encoding o given p': mismatched!, o = pc$_4$c$_5$, p'=c$_1$c$_2$c$_3$c$_4$.
(3) Encoding p' given p: p' = pc$_4$.
(4) Encoding o given p': matching, o = p'c$_5$.

Fig. 5. An example of the approach

See Figure 5, a selected pattern $p \in \mathcal{P}$ and a candidate $p' \in F$ overlap each other at $c_1c_2c_3$ on object o. Assume that o is encoded given p then $o = pc_4c_5$. As in Krimp algorithm, p' is still remained as origin and then p' cannot be used to encode o despite of p' occurs in o. This is because they are mismatched (i.e. $o = pc_4c_5, p' = c_1c_2c_3c_4$). To solve the problem, we propose to encode p' given p so that o and p' will contain the same pointer to p (i.e. $p' = pc_4$). Now, the regular encoding scheme can be applied to encode o given p' (i.e. $o = p'c_5$). We can consider that p and p' are overlapping but both of them can be included in the dictionary $\mathcal{P}$. **Note:** in our context, overlapped clusters are counted only once.

Main idea. Given a dictionary $\mathcal{P}$ and a chosen pattern p (i.e. will be added into $\mathcal{P}$), a set of pattern candidates F. The main idea is that we first encode the database O_{DB} given pattern p. Secondarily, we propose to encode all candidates $p' \in F$ given p in order to indicate the overlapping clusters between p and p'. After that, there are two kinds of pattern candidates which are encoded candidates and non-encoded candidates. Next, the best candidate in F will be put into $\mathcal{P}$ and used to encode O_{DB} and F. The process will be repeat until obtaining top-K patterns in the dictionary $\mathcal{P}$.

Let us consider the correlations between a pattern $p \in \mathcal{P}$ and a candidate $p' \in F$ to identify whenever encoding p' given p is needed. The correlation between p and p' is illustrated in Table 2. First of all, we do not allow overlap between two patterns of the same kind since they represent the same knowledge that may lead to extracting redundant information.

Table 2. Correlations between pattern p and pattern p' in F. O, $\varDelta$ and X respectively mean *"overlapping allowed, regular encoding"*, *"overlapping allowed, no encoding"* and *"overlapping not allowed"*.

		p		
		cs	$rGpattern^{\geq}$	$rGpattern^{\leq}$
p'	cs	X	O	O
	$rGpattern^{\geq}$	$\varDelta$	X	O
	$rGpattern^{\leq}$	$\varDelta$	O	X

Next, if p is a closed swarm then p' do not need to be encoded given p. This is because there are objects which contribute to gradual trajectories p' but not closed swarm. These objects cannot be encoded using p and therefore p' needs to be remained the same and the regular encoding scheme can be applied. Otherwise, p' will never be chosen later since there are no objects in O_{DB} which match p'. For instance, see Figure 2, the objects o_1 and o_4 do not contribute to the closed swarm p. Thus, if the gradual trajectory p' is encoded given p to indicate the overlapping clusters $c_5c_6c_7$ then that leads to a mismatched statement between o_1, o_4 and the gradual trajectory p'.

Until now, we already have two kinds of candidates $p' \in F$ (i.e. non-encoded and encoded candidates). Next, some candidates will be used to encode the database O_{DB}. To encode an object $o \in O_{DB}$ given a non-encoded candidate p', the regular encoding scheme mentioned in Section 3.1 can be applied. However, given an encoded candidate p', we need to perform an additional step before so that the encoding scheme can be applied regularly. This is because the two pointers referring to the same pattern $p \in \mathcal{P}$ from o (e.g. $[p, k]$) and from p' (e.g. $[p, l]$) can be different (i.e. $k \neq l$) despite the fact that p' is essentially included in o. That leads to a mismatched statement between o and p' and thus o cannot be encoded given p'.

For instance, see Figure 2, given a gradual trajectory pattern $rGpattern^{\geq}$ $p = c_3c_4c_5c_6c_7$, a closed swarm $p' = c_1c_2c_5c_6c_7c_9c_{10}$, the object $o_3 = c_1c_2c_4c_5c_6c_7c_9c_{10}$. We first encodes o_3 given p such that $o_3 = c_1c_2[p, 1]c_9c_{10}$. Then, p' is encoded given p, i.e. $p' = c_1c_2[p, 2]c_9c_{10}$. We can consider that the two pointers referring to p from o (i.e. $[p, 1]$) and from p' (i.e. $[p, 2]$) are different and thus o_3 and p' are mismatched. Therefore, o cannot be encoded given p' despite the fact that p' essentially occurs in o.

To deal with this issue, we simply recover uncommon clusters between the two pointers. For instance, to encode o_3 by using p', we first recover uncommon cluster such that $o_3 = c_1c_2c_4[p, 2]c_9c_{10}$. Note that $[p, 1] = c_4[p, 2]$. Since $p' = c_1c_2[p, 2]c_9c_{10}$, o_3 is encoded given p' such that $o_3 = p'c_4$.

Definition 5. *(Uncommon Clusters for rGpattern$^{\geq}$). Given a rGpattern$^{\geq}$, $p = c_1 \ldots c_n$ and two pointers refer to p, $[p, k]$ and $[p, l]$ with $k \leq l$. $uncom(p, k, l) = c_kc_{k+1} \ldots c_{l-1}$ is called an uncommon list of clusters between $[p, k]$ and $[p, l]$. Note that $[p, k] = c_kc_{k+1} \ldots c_{l-1}[p, l]$.*

Similarly, we also have $uncom(p, k, l)$ in the case p is a $rGpattern^{\leq}$. Until now, we are able to recover uncommon clusters between two pointers which refer to a pattern. Now, we start proving that given an object $o \in O_{DB}$ and a candidate $p' \in F$, if p' occurs in o then o can be encoded using p' even though they contain many pointers to other patterns. First, let us consider if p is a $rGpattern^{\geq}$ and p' is a closed swarm.

Lemma 1. *Given a rGpattern$^{\geq}$, $p = c_1 \dots c_n$, an object o and a closed swarm $p' \in F$. In general, if o and p' refer to p then $o = x_o[p,k]y_o$ and $p' = x_{p'}[p,l]y_{p'}$. Note that $x_o, y_o, x_{p'}$ and $y_{p'}$ are lists of clusters. If o contributes to p' then:*

$$k \leq l \wedge o = x_o\ uncom(p,k,l)[p,l]\ y_o \tag{4}$$

Proof. After construction if $k > l$ then $\exists c_i \in \{c_l, \dots, c_k\} (\subseteq p)$ s.t. $c_i \in p' \wedge c_i \notin o$. Therefore, o does not contribute to p' (Property 1). That suffers the assumption and thus we have $k \leq l$. Deal to the Definition 5, $[p,k] = uncom(p,k,l)[p,l]$. Consequently, we have $o = x_o\ uncom(p,k,l)[p,l]\ y_o$.

By applying Lemma 1, we have $o = x_o\ uncom(p,k,l)[p,l]\ y_o$ and $p' = x_{p'}[p,l]y_{p'}$. Then we can apply the regular encoding scheme to encode o given p'. let us assume that each object $o \in O_{p'}$ has a common list of pointers to other patterns as $\overrightarrow{(p',o)} = \big\{\big([p_1,l_1],[p_1,k_1]\big), \dots, \big([p_n,l_n],[p_n,k_n]\big)\big\}$ where $\forall i \in [1,n], [p_i,l_i]$ is the pointer from p' to p_i and $[p_i,k_i]$ is the pointer from o to p_i. If we respectively apply Lemma 1 on each pointer in $\overrightarrow{(p',o)}$ then o can be encoded given p'. Similarly, we also have the other lemmas for other pattern types.

Data description length computation. Until now, we have defined an encoding scheme for movement patterns. The description length of the dictionary in Table 1 is calculated as $L(\mathcal{P}) = |p_1|+1+|p_2|+1+|p_3|+1+|\mathcal{P}| = 3+1+4+1+2+1+2 = 14$. Similarly, description length of o_2 is $L(o_2|\mathcal{P}) = 1 + |[p_1,1]| + |[p_3,0]| + 1 = 6$.

Note: for each pattern, we need to consider an extra memory cell of pattern type. Additionally, for any given dictionary $\mathcal{P}$ and the data O_{DB}, the cost of storing the timestamp for each cluster is always constant regardless the size of the dictionary.

4 Mining Compression Object Movement Patterns

In this section we will present the two greedy algorithms which have been designed to extract a set of *top-K* movement patterns that compress the data best.

4.1 Naive Greedy Approach

The greedy approach takes as input a database O_{DB}, a candidate set F and a parameter K. The result is the optimal dictionary which encodes O_{DB} best. Now, at each iteration of *NaiveCompo*, we select candidate p' which compresses the database best. Next, p' will be added into the dictionary $\mathcal{P}$ and then the database O_{DB} and F will be encoded given p'. The process is repeated until we obtain K patterns in the dictionary.

To select the best candidate, we generate a duplication of the database O^d_{DB} and for each candidate $p' \in F$, we compress O^d_{DB}. The candidate p' which returns the smallest data description length will be considered as the best candidate. Note that $p' = argmin_{p^* \in F}\big(L_{p^*}(O_{DB})\big)$. The NAIVECOMPO is presented in Algorithm 1.

4.2 Smart Greedy Approach

Algorithm 1. NaiveCompo

```
Input  : Database $O_{DB}$, set of patterns $F$, int $K$
Output: Compressing patterns $\mathcal{P}$
Input  : Database $O_{DB}$, set of patterns $F$, int $K$
Output: Compressing patterns $\mathcal{P}$
begin
    $\mathcal{P} \longleftarrow \emptyset$;
    while $|\mathcal{P}| < K$ do
        foreach $p \in F$ do
            $O^d_{DB} \longleftarrow O_{DB}$;
            $L^*(O^d_{DB}|p) \longleftarrow$
            $CompressionSize(O^d_{DB}, p)$;
        $p^* \longleftarrow \arg\min_p L^*(O^d_{DB}|p)$;
        $\mathcal{P} \longleftarrow p^*$; $F \longleftarrow F \setminus \{p^*\}$;
        Replace all instances of $p^*$ in $O_{DB}$ by its pointers;
        Replace all instances of $p^*$ in $F$ by its pointers;
    output $\mathcal{P}$;
CompressionSize($O^d_{DB}, p$)
begin
    $size \longleftarrow 0$;
    foreach $o \in O_{DB}$ do
        if $p.involved(o) = true$ then
            Replace instance of $p$ in $o$ by its pointers;
    foreach $o \in O_{DB}$ do
        $size \longleftarrow size + |o|$;
    $size \longleftarrow size + |p| + 1$;
    output $size$;
```

The disadvantage of naive greedy algorithm is that we need to compress the duplicated database O^d_{DB} for each pattern candidate at each iteration. However, we can avoid this computation by considering some useful properties as follows.

Given a pattern p', $\overline{O}_{p'}$ and $O_{p'}$ respectively are the set of objects that do not contribute to p' and the set of objects involving in p'. The compression gain which is the number of memory cells we earned when adding p' into dictionary can be defined as $gain(p', \mathcal{P}) = L_{\mathcal{P}}(O_{DB}) - L_{\mathcal{P}\cup p'}(O_{DB})$.

The fact is that we can compute the compression gain by scanning objects $o \in O_{p'}$ with p'. Each pattern type has its own compression gain computation function. Let us start presenting the process by proposing the property for a closed swarm p'.

Property 2. Given a dictionary $\mathcal{P}$, a closed swarm $p' \in F$. $gain(p', \mathcal{P})$ is computed as:

$$gain(p', \mathcal{P}) = |O_{p'}| \times |p'| - \Big(\sum_{o}^{O_{p'}} \sum_{i}^{\overrightarrow{(p', o)}} |l_i - k_i| + |p'| + |O_{p'}| + 2 \Big) \qquad (5)$$

Proof. After construction we have $L_{\mathcal{P}\cup p'}(O_{DB}) = L(\mathcal{P} \cup p') + L(O_{DB}|\mathcal{P} \cup p') = (L(\mathcal{P}) + |p'| + 2) + L(\overline{O}_{p'}|\mathcal{P}) + L(O_{p'}|\mathcal{P} \cup p')$. Note that $L(\overline{O}_{p'}|\mathcal{P}) = L(\overline{O}_{p'}|\mathcal{P} \cup p')$. Furthermore, $\forall o \in O_{p'} : L(o|\mathcal{P} \cup p') = L(o|\mathcal{P}) - |p'| + 1 + \sum_i^{\overrightarrow{(p', o)}} |l_i - k_i|$. Thus, $L(O_{p'}|\mathcal{P}\cup p') = \sum_{o \in O_{p'}} L(o|P \cup p') = L(O_{p'}|\mathcal{P}) - |O_{p'}| \times |p'| + \sum_o^{O_{p'}} \sum_i^{\overrightarrow{(p', o)}} |l_i - k_i| + |O_{p'}|$. Therefore, we have $L_{\mathcal{P}\cup p'}(O_{DB}) = L(\mathcal{P}) + L(\overline{O}_{p'}|\mathcal{P}) + L(O_{p'}|\mathcal{P}) - |O_{p'}| \times |p'| + \big(\sum_o^{O_{p'}} \sum_i^{\overrightarrow{(p', o)}} |l_i - k_i| + |p'| + |O_{p'}| + 2 \big)$. Note that $L(O_{DB}|\mathcal{P}) = L(\overline{O}_{p'}|\mathcal{P}) + L(O_{p'}|\mathcal{P})$. Consequently, we have $gain(p', \mathcal{P}) = |O_{p'}| \times |p'| - \big(\sum_o^{O_{p'}} \sum_i^{\overrightarrow{(p', o)}} |l_i - k_i| + |p'| + |O_{p'}| + 2 \big)$.

By applying Property 2, we can compute the compression gain when adding a new closed swarm p' into the dictionary $\mathcal{P}$. In the Equation 5, the compression $gain(p', \mathcal{P})$

depends on the size of p', $O(p')$ and the number of uncommon clusters that can be computed by scanning p' with objects $o \in O(p')$ without encoding O_{DB}. Due to the space limitation, we will not describe properties and proofs for the other pattern types (i.e. $rGpattern^{\geq}$, $rGpattern^{\leq}$) but they can be easily derived in a same way as Property 2.

To select the best candidate at each iteration, we need to chose the candidate which returns the best compression gain. SmartCompo is presented in the Algorithm 2.

5 Experimental Results

Algorithm 2. SmartCompo

```
Input  : Database $O_{DB}$, set of patterns $F$, int $K$
Output: Compressing patterns $\mathcal{P}$
begin
    $\mathcal{P} \longleftarrow \emptyset$;
    while $|\mathcal{P}| < K$ do
        foreach $p \in F$ do
            $L^*(O_{DB}|p) \longleftarrow Benefit(O_{DB}, p)$;
        $p^* \longleftarrow \arg\min_p L^*(O_{DB}|p)$;
        $\mathcal{P} \longleftarrow p^*$; $F \longleftarrow F \setminus \{p^*\}$;
        Replace all instances of $p^*$ in $O_{DB}$ by its pointers;
        Replace all instances of $p^*$ in $F$ by its pointers;
    output $\mathcal{P}$;
Benefit($O^d_{DB}, p$)
begin
    $b \longleftarrow 0$;
    foreach $o \in O_{DB}$ do
        if $p.involved(o) = true$ then
            $b \longleftarrow b + benefit(o, p)$;
    $b \longleftarrow b + |p| + 1$;
    output $b$;
```

A comprehensive performance study has been conducted on real-life 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++ 4.6.1.

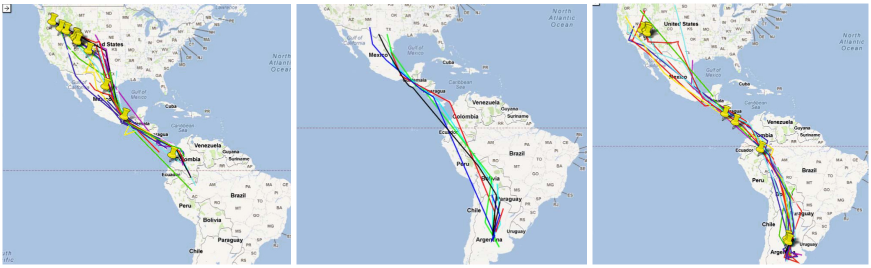
As in [10] [6], the two following datasets[1] have been used during experiments: Swainsoni dataset includes 43 objects evolving over 764 different timestamps. The dataset was generated from July 1995 to June 1998. Buffalo dataset concerns 165 buffaloes and the tracking time from year 2000 to year 2006. The original data has 26610 reported locations and 3001 timestamps. Similarly to [7] [10], we first use linear interpolation to fill in the missing data. Furthermore, DBScan [2] ($MinPts = 2; Eps = 0.001$) is applied to generate clusters at each timestamp. In the comparison, we compare the set of patterns produced by SmartCompo with the set of closed swarms extracted by *ObjectGrowth* [10] and the set of gradual trajectories extracted by *ClusterGrowth* [6].

Effectiveness. We compare the top-5 highest support closed swarms, the top-5 highest covered area gradual trajectory patterns and the top-5 compression patterns from Swainsoni dataset. Each color represents a Swainsoni trajectory involved in the pattern.

Top-5 closed swarms are very redundant since they only express that Swainsonies move together from North America to Argentina. Similarly, top-5 rGpatterns are also redundant. They express the same knowledge that is *"from 1996-10-01 to 1996-10-25, the more time passes, the more objects are following the trajectory* $\{Oregon\rangle\ Nevada\rangle\ Utah\rangle\ Arizona\rangle\ Mexico\rangle\ Colombia\}$*"*.

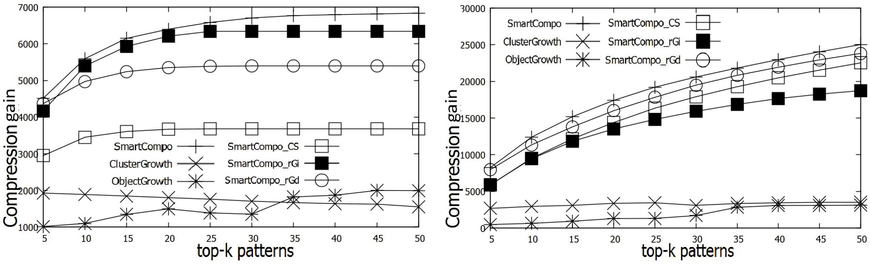
Figure 6 illustrates 3 patterns among 5 extracted ones by using SmartCompo. The $rGpattern^{\geq}$ expresses the same knowledge with the mentioned rGpattern in the top highest covered area. The closed swarm expresses new information that is *"after arriving South America, the Swainsonies tend to move together to Argentina even some*

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



(a) $rGpattern \geq$ (b) Closed swarm (c) $rGpattern^{\leq}$

Fig. 6. Top-3 typical compression patterns



(a) Swainsoni dataset (b) Buffalo dataset

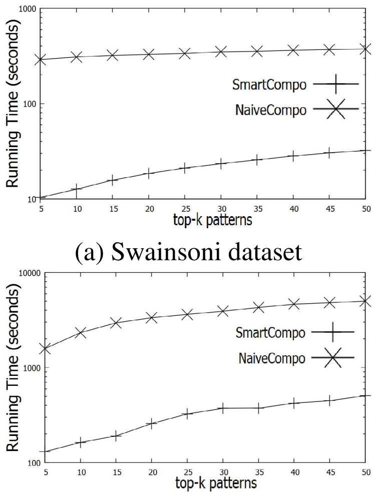
Fig. 7. Compressibility (higher is better) of different algorithms

of them can leave their group". Next, the $rGpattern^{\leq}$ shows that *"the Swainsonies return back together to North America from Argentina (i.e. 25 objects at Argentina) and they will step by step leave their group after arriving Guatemala (i.e. 20 objects at Guatemala) since they are only 2 objects at the last stop, i.e. Oregon State"*.

Compressibility. We measure the compressibility of the algorithms by using their *top*-K patterns as dictionaries for encoding the data. Since NaiveCompo and SmartCompo provides the same results, we only show the compression gain of SmartCompo.

Regarding to SmartCompo, the compression gain could be calculated as the sum of the compression gain returned after each greedy step with all kinds of patterns in F. For each individual pattern type, compression gain is calculated according to the greedy encoding scheme used for SmartCompo. They are respectively denoted as $SmartCompo\_CS$ (i.e. for closed swarms), $SmartCompo\_rGi$ (i.e. for $rGpattern^{\geq}$) and $SmartCompo\_rGd$ (i.e. for $rGpattern^{\leq}$). Additionally, to illustrate the difference between MDL-based approaches and standard support-based approaches, we also employ the set of *top*-K highest support closed swarms and *top*-K highest covered area gradual trajectories patterns.

Figure 7 shows the compression gain of different algorithms. We can consider that *top*-K highest support or covered area patterns cannot provide good compression gain since they are very redundant. Furthermore, if we only consider one pattern type, we cannot compress the data best since the compression gains of SmartCompo_CS, SmartCompo_rGi and SmartCompo_rGd are always lower than SmartCompo. This is because the pattern distribution in the data is complex and different patterns can cover different parts of the data. Thus, considering one kind of patterns results in losing interesting patterns and not good compression gain. By proposing overlapping allowed multi-pattern structure encoding scheme, we are able to extract more informative patterns.



(a) Swainsoni dataset

(b) Buffalo dataset

Fig. 8. Running time

One of the most interesting phenomena is that the Swainsonies and Buffaloes have quite different movement behavior. See Figure 7a, we can consider that $rGpattern^{\geq}$ is the most representative movement behavior of Swainsonies since they compress the data better than the two other ones. While closed swarm is not as representative as the other patterns. This is because it is very easy for Swainsonies which are birds to leave the group and congregate again at later timestamps. However, this movement behavior is not really true for Buffaloes. See Figure 7b, it clear that the compression gains of closed swarms, $rGpattern^{\geq}$ and $rGpattern^{\leq}$ have changed. The three kinds of patterns have more similar compression gain than the ones in Swainsonies. It means that Buffaloes are more closed to each other and they move in a dense group. Thus closed swarm is more representative compare to itself in Swainsoni dataset. Furthermore, the number of Buffaloes is very difficult to increase in a group and thus SmartCompo_rGi is lower than the two other ones.

Running Time. In our best knowledge, there are no previous work which address mining compression movement pattern issue. Thus, we only compare the two proposed approaches in order to highlight the differences between them. Running time of each algorithm is measured by repeating the experiment in compression gain experiment.

As expected, SmartCompo is much faster than NaiveCompo (i.e. Figure 8). By exploiting the properties, we can directly select the best candidate at each iteration. Consequently, the process efficiency is speed up.

6 Related Work

Mining informative patterns can be classified into 3 main lines: MDL-based approaches, statistical approaches based on hypothesis tests and information theoretic approaches.

The idea of using data compression for data mining was first proposed by R. Cilibrasi et al. [1] for data clustering problem. This idea was also explored by Keogh et al. [8], who propose to use compressibility as a measure of distance between two sequences. In the second research line, the significance of patterns is tested by using a standard statistical hypothesis assuming that the data follows the null hypothesis. If a pattern pass the test it is considered significant and interesting. For instance, A. Gionis et al. [3] use swap randomization to generate random transactional data from the original data.

A similar method is proposed for graph data by R. Milo et al. [11]. Another research direction looks for interesting sets of patterns that compress the given data most (i.e. MDL principle). Examples of this direction include the Krimp algorithm [14] and Slim algorithm [12] for itemset data and the algorithms for sequence data [9].

7 Conclusion

We have explored an MDL-based strategy to compress moving object data in order to: 1) select informative patterns, 2) combine different kinds of movement patterns with overlapping allowed. We supplied two algorithms NaiveCompo and SmartCompo. The latter one exploits smart properties to speed up the whole process obtaining the same results to the naive one. Evaluations on real-life datasets show that the proposed approaches are able to compress data better than considering just one kind of patterns.

References

1. Cilibrasi, R., Vitányi, P.M.B.: Clustering by compression. IEEE Transactions on Information Theory 51(4), 1523–1545 (2005)
2. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, pp. 226–231 (1996)
3. Gionis, A., Mannila, H., Mielikäinen, T., Tsaparas, P.: Assessing data mining results via swap randomization. TKDD 1(3) (2007)
4. Grunwald, P.: The minimum description length principle. The MIT Press (2007)
5. Gudmundsson, J., van Kreveld, M.: Computing longest duration flocks in trajectory data. In: ACM GIS 2006 (2006)
6. Hai, P.N., Ienco, D., Poncelet, P., Teisseire, M.: Ming time relaxed gradual moving object clusters. In: ACM SIGSPATIAL GIS (2012)
7. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. Proc. VLDB Endow. 1(1), 1068–1080 (2008)
8. Keogh, E.J., Lonardi, S., Ratanamahatana, C.A., Wei, L., Lee, S.H., Handley, J.: Compression-based data mining of sequential data. DMKD 14(1), 99–129 (2007)
9. Lam, H.T., Moerchen, F., Fradkin, D., Calders, T.: Mining compressing sequential patterns. In: SDM, pp. 319–330 (2012)
10. Li, Z., Ding, B., Han, J., Kays, R.: Swarm: mining relaxed temporal moving object clusters. Proc. VLDB Endow. 3(1-2), 723–734 (2010)
11. Milo, R., Shen-Orr, S., Itzkovits, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: Simple building blocks of complex networks. Science 298(5594) (2002)
12. Smets, K., Vreeken, J.: Slim: Directly mining descriptive patterns. In: SDM (2012)
13. Tang, L.A., Zheng, Y., Yuan, J., Han, J., Leung, A., Hung, C.-C., Peng, W.-C.: On discovery of traveling companions from streaming trajectories. In: ICDE, pp. 186–197 (2012)
14. Vreeken, J., Leeuwen, M., Siebes, A.: Krimp: Mining itemsets that compress. Data Min. Knowl. Discov. 23(1), 169–214 (2011)
15. Wang, Y., Lim, E.P., Hwang, S.Y.: Efficient mining of group patterns from user movement data. Data Knowl. Eng. 57(3), 240–282 (2006)
16. Zheng, K., Zheng, Y., Yuan, J., Shang, S.: On discovery of gathering patterns from trajectories. In: ICDE (2013)