ACADÉMIE DE MONTPELLIER
UNIVERSITÉ **M**ONTPELLIER **2**
Sciences et **T**echniques du **L**anguedoc

PH.D THESIS

présentée au Laboratoire d'Informatique de Robotique
et de Microélectronique de Montpellier pour
obtenir le diplôme de doctorat

Spécialité	:	**Informatique**
Formation Doctorale	:	**Informatique**
École Doctorale	:	**Information, Structures, Systèmes**

Mining Object Movement Patterns from Trajectory Data

par

PHAN NHAT HAI

Version du September 30, 2013

Supervisor
Mme. Maguelonne TEISSEIRE, Research Director ..... IRSTEA, Montpellier

Joint supervisor
M. Pascal PONCELET, Professor ..... LIRMM, Université Montpellier II

Reviewers
M. Osmar ZAÏANE, Professor ..... University of Alberta
M. Arno SIEBES, Professor ..... Utrech University

Examinators
M. Francesco BONCHI, Senior Researcher ..... Yahoo! Research Lab
M. Bruno CREMILLEUX, Professor ..... Université de Caen
M. Dino IENCO, Researcher ..... IRSTEA, Montpellier

First and foremost I want to thank my advisors: Dr. Dino Ienco, Prof. Pascal Poncelet and Dr. Maguelonne Teisseire. It has been an honor to be their PhD student. I appreciate all their valuable support of time, ideas, and funding to make my PhD experience productive and stimulating. The joy and enthusiasm they have for their research was contagious and motivational for me, even during tough times in the PhD pursuit. The members of the Tatoo team have contributed immensely to my personal and professional time at Irstea, Lirmm, and University of Montpellier 2. The team has been a source of friendships as well as good advice and collaboration. I am especially grateful for the fun team members who stuck it out with me during my PhD duration: Hugo Alatrista-Salas, Mickaël Fabrègue, and Flavien Bouillot. We worked together on the mining trajectories on Tweets experiments, and I very much appreciated his enthusiasm, intensity, willingness to do all the tasks. In additionally, I am also thankful to Amal Zine El Aabidine for her help in mining trajectories from gene data source. I thank Prof. Donato Malerba (University of Bari) for not only proposing the multi-relational gradual pattern idea but also advising me on the very interesting work. Under his support, we are able to extract an novel kind of gradual patterns based on two different point of views. In my later work of Communication Graph Compression, I am grateful to Dr. Francesco Bonchi (Senior Researcher at Yahoo! Lab) for his support, guidance and all fun along during three months of my internship at Yahoo! Research Lab, Barcelona. I gratefully acknowledge the funding sources that made my PhD work possible. I was funded by the French National Centre for Scientific Research (CNRS). My work was also supported by the LIRMM - IRSTEA Laboratories. For this dissertation I would like to thank my reading committee members: Prof. Osmar Zaiane, Prof. Arno Seibes, Dr. Francesco Bonchi, and Prof. Bruno Cremilleux for their time, interest, helpful

comments, and insightful questions. My time at Montpellier was made enjoyable in large part due to the many friends and groups that became a part of my life. I am grateful for time spent with roommates and friends, for my backpacking buddies and our memorable trips into the mountains, beaches, seas, and ancient cities in Europe as I finished up my degree, and for many other people and memories. Lastly, I would like to thank my family for all their love and encouragement. For my parents who raised me with a love of science and supported me in all my pursuits. And most of all for my loving, supportive, encouraging, and patient girl friend Huong whose faithful support during the final stages of this PhD is so appreciated. Thank you. PHAN Nhat Hai, Irstea-Lirmm Labs, University Montpellier 2, October 2013.

Contents

List of Figures

List of Tables

CHAPTER

1 Introduction

With the rapid development of positioning technologies, sensor networks, and online social media, spatio-temporal data is now widely collected from smartphones carried by people, sensor tags attached to animals, GPS tracking systems on cars and airplanes, RFID tags on merchandise, and location-based services offered by social media. Indeed, analyzing spatio-temporal data generated from these systems frames many research problems and high-impact applications:

. Understanding animal movement is important to addressing environmental challenges such as climate and land use change, bio-diversity loss, invasive species, and infectious diseases.
. Traffic patterns help people understand conditions of road networks and better design future transportation systems; analyzing driving patterns combining with weather conditions could improve routing systems.
. Unusual vessel trajectory could be a sign of smuggling; outlying taking-off/landing patterns could be a dangerous signal for aviation; and detection of suspicious human movements could help prevent crimes and terrorism.

Motivated by the huge benefit of many applications, a lot of researchers have been focusing their research motivation to spatio-temporal data mining. One of the key techniques is to analyze such datasets for meaningful patterns, called moving object clusters [7] [17] [18] [23]. A moving object cluster can be defined in both spatial and temporal dimensions: (1) a group of moving objects should be geometrically close to each other, (2) they should be together for at least min_t timestamps. In this context, many recent studies have been conducted to mine moving object clusters including flocks [7] [30] [39], moving clusters [19] [17], convoy queries [18] [2], k-star [29], closed swarms [23] [9], traveling companions [28], group patterns [32], periodic patterns [24], etc. To extract these kinds of patterns, many algorithms have been proposed such as CuTS* (convoy mining),

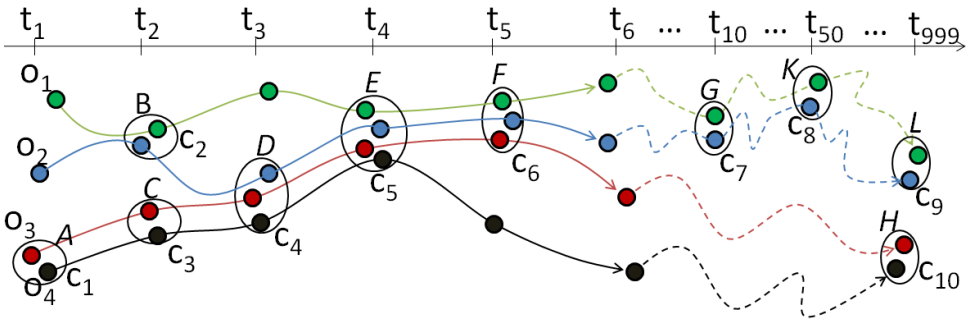


Figure 1.1: A running example of moving object data.

ObjectGrowth (closed swarm mining), VG-Growth (group pattern mining), BFE (flock mining), Periodica (periodic pattern mining), etc. Interested readers may refer to [14] where descriptions of the most efficient approaches and patterns are presented.

For instance, let us consider a database in which there are four cars moving during the time and their locations are reported by GPS system. We have 4 cars moving from t_1 to t_{999}, $\forall i \in [1:10], c_i$ is a cluster generated by applying clustering technique as presented in Figure 1.1. In most of the chapters, we will use (a part of) this running example for enhancing some particularities that are addressed in the chapter. One of the extracted convoys can be *"the two cars,* o_3 *and* o_4*, are consecutively moving together from* t_1 *to* t_4*"* or a potential closed swarm is *"the two cars,* o_1 *and* o_2*, are moving together from* t_2 *to* t_{999}*, from B to L"*, etc. Even though these kinds of patterns are meaningful, there are some challenging issues that limit their utility.

1.1 Illustrative Example and Motivations

In this section, we will give an example to generally present challenging issues and our motivations in object movement pattern mining.

1) The first issue is how to efficiently manage different kinds of patterns? Indeed, given a specific dataset, e.g. see Figure 1.1, it is difficult to know which kind of patterns is useful to analyze the data. Intuitively, that could be a set of closed swarms indicate that the cars o_1 and o_2 or the cars o_3 and o_4 are moving together. However, that also could be a set of moving object clusters state that the cars o_1, o_2, o_3 and o_4 are moving together from C to D to E to F, etc.

To deal with this issue, one of potential solutions is employing all the existing approaches, each of which extracts a specific pattern, to obtain all kinds of patterns. Then, user can analyze the final results to understand the object movement behavior. Naturally, the computation is costly and time consuming since we need to execute different algorithms consecutively. Additionally, in some applications object locations are continuously

updated, e.g. cars report their locations by using Global Positioning System (GPS). Therefore, new data is always available and it means that we need to execute again and again the algorithms on the entire data to update the final results. This is of course, cost-prohibitive and time consuming as well.

2) The second issue is that the existing patterns are relevant and help us to fully understand the complex and unpredictable behavior of moving objects or not. For instance, they require the group of moving objects to be together for at least min_t timestamps, i.e. could be consecutive or completely be non-consecutive, which might not be practical in the real cases. Enforcing consecutive time constraint may results in loosing meaningful patterns while completely relax this constrain may generate large amount of extraneous patterns.

For instance, see Figure 1.1, enforcing the consecutive time constraint results in losing of the pattern *"the two cars,* o_1 *and* o_2*, are moving together from* B *to* E *and to* F*"* since they are not close each other at time t_3. While completely relax this constraint will generate the extraneous patterns such as *"the two cars,* o_1 *and* o_2*, are moving together from* B *to* E *to* F *to* G *to* K *and to* L*"*. This pattern are irrelevant because they meet each other at L after 949 timestamps by chance and not actually moving together from K to L.

Another illustrative example is that the traditional movement patterns usually focus on an unchanged group of objects and thus they cannot capture the object moving trend. Indeed, objects can step by step getting together to go to some place or leaving each other from that place.

For instance, in Figure 1.1, *"all the cars are gathering at* E *after* A *to* C *and to* D*"*. This phenomenon is involved in many real world applications such as traffic congestion, animal or population migration, location prediction, etc. Therefore, it demands the definition of novel movement pattern models to enrich the application benefit.

3) Naturally, end user can be overwhelmed by a huge number of extracted patterns although only a few of them are useful. Indeed, thousand of patterns representing redundant knowledge clearly poses limit in their usefulness. However, relatively few researchers have addressed the problem of reducing movement pattern redundancy.

4) The potential utility of the developed spatio-temporal mining tools must be justified in their applicable field. One of our ultimate goals is to develop techniques that not only benefit the computer science society, but more importantly, are applicable to various interdisciplinary science and engineering fields. Therefore, it is always crucial for us to emphasize the practicability of our methods. Performance measures of the developed tools in terms of accessibility, scalability, reliability and other human factors also need to be carefully studied, in order to provide answers to concerns from domain experts.

1.2 Contributions

All of the mentioned issues have been addressed in my thesis. We propose a three step framework, i.e. Figure 1.2. First of all, we develop a unifying approach to extract and man-

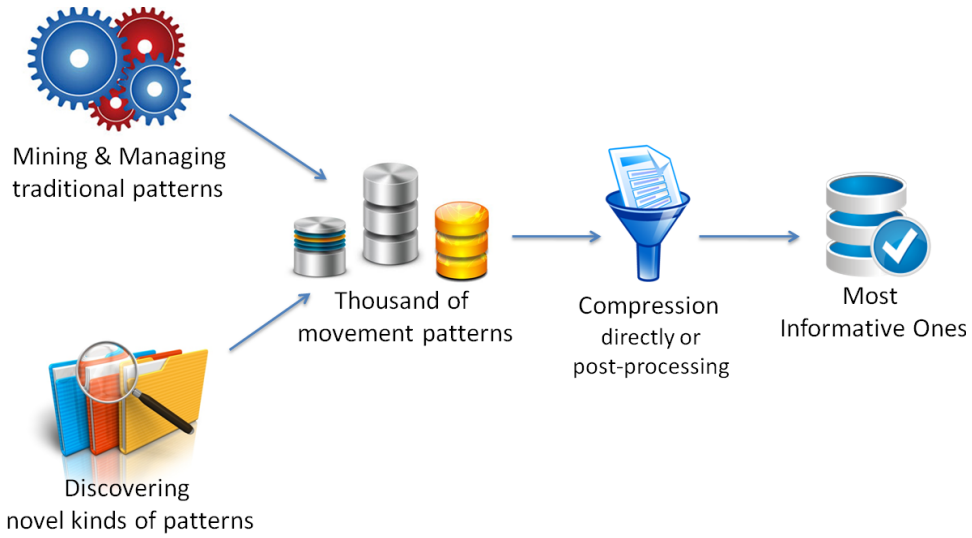


Figure 1.2: Our three step framework on mining and managing object movement patterns.

age multiple patterns. Second, we discover novel kinds of movement patterns and efficient algorithms to mine them. To select informative patterns from thousand of very redundant ones, we further design a compression step based on Minimum Description Length principal. To justify the utility of the proposed concepts and developed approaches, we also supply a demonstration system, named MULTI_MOVE, which is designed to efficiently and automatically extract different movement patterns. Our contributions are generally presented as follows:

All in One: Mining Multiple Movement Patterns. Designing a unifying approach which can manage multiple movement patterns is very challenging. This is because: 1) there are many kinds of patterns and each of which have their own characteristics. 2) Additionally the results need to be reusable to obtain the final results when new data is available. 3) Obtaining the optimal parameters is a difficult task for most of algorithms which require parameter setting.

All these challenging issues have been throughly considered and well addressed in my proposed method, GET_MOVE [12]. The basic idea of GET_MOVE is to reconfigure moving object data to a *cluster matrix* then movement patterns will be redefined in an itemset context. Next, frequent closed itemset will be adopted to efficiently extract closed itemsets from which movement patterns can be mined.

Fuzzy Moving Object Clusters. To soften the consecutive time constraint, the key challenge is to deal with the time gap between a pair of clusters since: 1) it is difficult to recognize which size of a time gap is relevant or not, 2) we need to know when the patterns should be ended to eliminate uninteresting ones.

To address these issues, we present the definition of fuzzy time gap and fuzzy time

gap participation index. Obtained patterns are of the type *"the two cars,* o_1 *and* o_2*, are moving together from B to E to E to G and to K with 60% weak, 20% medium and 20% strong time gaps"*, see Figure 1.1. These patterns are characterized by their time gap frequency (or support), which is by definition the proportion of time gaps involved in the patterns. To extract all the fuzzy closed swarms, we propose a novel itemset-based property so that GET_MOVE can be employed to gather all the fuzzy closed swarms.

Time Relaxed Gradual Trajectory Patterns. A gradual moving object cluster is a list of moving object clusters which satisfy the graduality constraint and integrity condition during at least $\mathtt{min}_t$ 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.

As exemplified in Figure 1.1, the retrieved knowledge from traditional patterns can be *"the two cars,* o_1 *and* o_2*, are moving together from* t_2 *to* t_6*"* or *"the two cars,* o_3 *and* o_4*, are moving together from* t_1 *to* t_4*"*. Even if these patterns are relevant, they do not really present the actual moving behavior which can be *"from* t_1 *to* t_4*, as time passes, the more cars are following the trajectory* $\{A \rangle\ C \rangle\ D \rangle\ E\}$*"*.

To avoid finding redundant rGpatterns, we only focus on discovering the complete set of *maximal rGpatterns*. The basic idea is that if C is a rGpattern, it is unnecessary to output any subset C' of C even if C' may also satisfy rGpattern requirements.

Efficiently extracting of complete set of maximal rGpatterns in a large moving object data, denoted db, is a non-trivial task. First, the size of all possible combinations is exponential, i.e. approximately $2^{|C_{db}|}$ where C_{db} is the set of all clusters extracted from the data db and $|C_{db}|$ is significantly large. Second, none of previous work (i.e. frequent pattern mining [1] [15], moving object clusters [7] [17] [18] [23]) solves exactly the same issue as finding maximal rGpatterns. This is because they do not address the graduality in terms of itemsets or moving clusters. However, graduality is a key feature in rGpattern context. Thus, the discovery of rGpatterns introduces a new problem that needs to be solved by specifically designed techniques.

Facing the huge potential search space, we propose an efficient approach, named CLUSTERGROWTH, which shares the same spirit with the ObjectGrowth algorithm [23] but different in terms of design and goal. In CLUSTERGROWTH, we design two efficient rules which are *Graduality Pruning* and *Backward Pruning* to end unnecessary further search. After cutting a great portion of invalid candidates through the pruning rules, we perform a further check, i.e. *Actual Maximum Checking*, to report *interesting maximal rGpatterns* on-the-fly. This final step avoids using more space to store candidates and extra time for post-processing.

To enrich the utility of the rGpattern concept, we adapt the Minimum Description Length (MDL) principle for mining representative rGpatterns. An encoding scheme which is designed to deal with different kinds of overlapping rGpattern structures is proposed. We show that mining representative rGpatterns is NP-Hard and therefore we propose two heuristic algorithms to extract compressing rGpatterns. The first algorithm, named COM-

POGP, uses a greedy two-phase approach. To overcome performance with the required candidate generation in COMPOGP we propose an effective algorithm, called DICOMPOGP, to directly mine compressing rGpatterns.

Mining Representative Object Movement Patterns. 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 type of moving object 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 core of the method is to design and encoding scheme for different movement patterns, and then an approach which allow overlapping between them. The proposed method results in a rank of the patterns involved in the summarization of the dataset.

MULTI_MOVE **System.** Despite the growing demands for diverse applications, there have been few scalable tools for mining massive and sophisticated moving object data. Even if some tools are available for extracting patterns (e.g. [33]), they mainly focus on specific kinds of patterns at a time. Obviously, when considering a dataset, it is quite difficult for the decision maker to know in advance which kinds of patterns are embedded in the data. To cope with this issue, we propose the MULTI_MOVE system to reveal, automatically and in a very efficient way, collective movement patterns like convoys, group patterns, closed swarms, moving clusters and also periodic patterns. Starting from the results of MULTI_MOVE, the user can then visualize, browse and compare the different extracted patterns through a user friendly interface in Google Maps and Google Earth. The MULTI_MOVE system has gained a lot of publicity, since it is the first system that enable users to obtain the comparative results.

Moreover, we also demonstrate that our proposed approaches and systems can be efficiently applied to many other domains such as *gene expression analysis*, and *tweet user behavior analyzing*. The GET_MOVE approach was applied on 3 HIV time-series gene expression dataset to outline relationships between genes based on their expressions at different timestamps following infection. We have found that clustering gene expression data groups together efficiently genes of similar function based on their FC value and coherent results with our knowledge on HIV-1 versus HIV-2 infection were obtained. In additional, trajectories expressing the evolution of *French political communities* can be extracted from the political tweets which have been gathered during the *French Election Campaign 2012*.

Mining Multi-Relational Gradual Patterns. As an extra work, we also present the *multi-relational gradual patterns*. Indeed, gradual patterns highlight covariations of attributes of the form *"The more/less X, the more/less Y"*. Their usefulness in several applica-

tions has recently stimulated the synthesis of several algorithms for their automated discovery from large datasets. However, existing techniques require all the interesting data to be in a single database relation or table. This work extends the notion of gradual pattern to the case in which the co-variations are possibly expressed between attributes of different database relations. The interestingness measure for this class of “relational gradual patterns" is defined on the basis of both Kendall's τ and gradual support. Moreover, we propose two algorithms, named τRGp and gradRGp, for the discovery of relational gradual rules, and three pruning strategies to reduce the search space. The efficiency of the algorithms is empirically validated, and the usefulness of relational gradual patterns is proved on some real-world databases.

The remaining of the report is organized as follows. The related work is discussed in Chapter 2. GET_MOVE is presented in Chapter 3. Mining fuzzy closed swarms and gradual trajectory patterns are described in Chapters 4 and 5 separately. Mining representative movement patterns will be introduced in Chapter 6. I also briefly present MULTI_MOVE system in Chapter 7. The extra work, i.e. mining multi-relational gradual patterns, and the conclusion will be drawn in Chapters 8 and 9.

CHAPTER

2 Related Work

Preamble

The problem of object movement patterns has been extensively addressed over the last years. Basically, an object movement patterns are designed to group similar trajectories or objects which tend to move together during a time interval. So many different definitions can be proposed and today lots of patterns have been defined such as flocks, convoys, closed swarms, moving clusters and even periodic patterns.

2.1 Preliminary Definitions

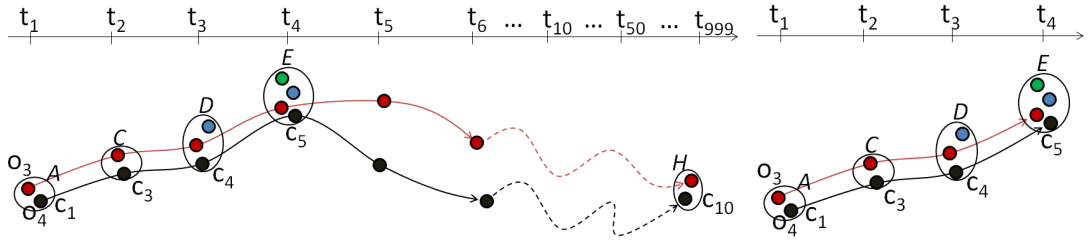
First of all, let us assume that we have a group of moving objects $O_{db} = \{o_1, o_2, \ldots, o_z\}$, a set of timestamps $T_{db} = \{t_1, t_2, \ldots, t_n\}$ and at each timestamp $t_i \in T_{db}$, spatial information [1] x, y for each object. For example, Table 2.1 illustrates an example of a moving object database. Usually, in object movement pattern 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 min_o be the user-defined threshold standing for a minimum number of objects and min_t the minimum number of timestamps. Thus $|O|$ (resp. $|T|$) must be greater than or equal to min_o (resp. min_t).

Database of clusters. A database of clusters, $C_{db} = \{C_{t_1}, C_{t_2}, \ldots, C_{t_n}\}$, is the collection of snapshots of the moving object clusters at timestamps $\{t_1, t_2, \ldots, t_n\}$. Note that an object could belong to several clusters at one timestamp (i.e. cluster overlapping). Given a cluster $c \in C_t (\in C_{db})$ and $c \subseteq O_{db}$, $|c|$ and $t(c)$ are respectively used to denote the number of

1. As illustrated in the introduction, spatial information can be for instance GPS location.

Table 2.1: An example of a moving object 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
o_2	t_2	0.3	1.2



(a) Swarm (b) Convoy

Figure 2.1: An example of swarm and convoy extracted from the Figure 1.1.

objects belong to cluster c and the timestamp that c involved in. To make the process more general, the clustering [2] is taken as a preprocessing step. In the following, we formally define all the different kinds of movement patterns.

Informally, a *swarm* is a group of moving objects O containing at least $\mathtt{min}_o$ individuals which are closed each other for at least $\mathtt{min}_t$ timestamps. Then a swarm can be formally defined as follows:

Definition 1. *Swarm[23]. A pair* (O,T) *is a swarm if:*

$$\begin{cases} (1): \forall t_{a_i} \in T, \exists c \text{ s.t. } O \subseteq c, \text{ c is a cluster.} \\ (2): |O| \geq \mathtt{min}_o. \\ (3): |T| \geq \mathtt{min}_t. \end{cases} \tag{2.1}$$

Note that the meaning of the above conditions are: (1) there is at least one cluster containing all the objects in O *at each timestamp in* T*, (2) there must be at least* $\mathtt{min}_o$ *objects, (3) there must be at least* $\mathtt{min}_t$ *timestamps.*

2. The clustering method is not fixed in our system. Users can cluster cars along highways using a density-based method, or cluster birds in 3 dimension space using the k-means algorithm. Clustering methods that generate overlapping clusters are also applicable, such as EM algorithm or using a rigid definition of the radius to define a cluster. Moreover, clustering parameters are decided by users' requirements or can be indirectly controlled by setting the number of clusters at each timestamp.

Usually, most of clustering methods can be done in polynomial time. In our experiments, we used DB-Scan [4], which takes $O(|O_{db}|\log(|O_{db}|) \times |T_{db}|)$ in total to do clustering at every timestamp. To speed it up, there are also many incremental clustering methods for moving objects. Instead computing clusters at each timestamp, clusters can be incrementally updated from last timestamps.

For example, as shown in Figure 2.1a, if we set $min_o = 2$ and $min_t = 2$, we can find the following swarms $(\{o_3, o_4\},\{t_1, t_2\})$, $(\{o_3, o_4\},\{t_1, t_2, t_3\})$, $(\{o_3, o_4\},\{t_1, t_3\})$, $(\{o_3, o_4\},\{t_1, t_2, t_3, t_4, t_{999}\})$, etc. We can note that these swarms are in fact redundant since they can be grouped together in the following swarm $(\{o_3, o_4\},\{t_1, t_2, t_3, t_4, t_{999}\})$.

To avoid this redundancy, Zhenhui Li et al. [23] propose the notion of *closed swarm* for grouping together both objects and timstamps. 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 and can be defined as follows:

Definition 2. *Closed Swarm [23]. A pair* (O, T) *is a closed swarm if:*

$$\begin{cases} (1) : (O, T) \textit{ is a swarm.} \\ (2) : \nexists O' \textit{ s.t. } (O', T) \textit{ is a swarm and } O \subset O'. \\ (3) : \nexists T' \textit{ s.t. } (O, T') \textit{ is a swarm and } T \subset T'. \end{cases} \tag{2.2}$$

For instance, in the previous example, $(\{o_3, o_4\},\{t_1, t_2, t_3, t_4, t_{999}\})$ is a closed swarm.

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 swarm (or closed swarm) is that convoy lifetimes must be consecutive. In essential, by adding the consecutiveness condition to swarms, we can define convoy as follows:

Definition 3. *Convoy [18]. A pair* (O, T)*, is a convoy if:*

$$\begin{cases} (1) : (O, T) \textit{ is a swarm.} \\ (2) : \forall i, 1 \leq i < |T|, t_{a_{i+1}} = t_{a_i} + 1. \end{cases} \tag{2.3}$$

For instance, on Figure 2.1b, with $min_o = 2, min_t = 2$ we have a convoy $(\{o_3, o_4\},\{t_1, t_2, t_3, t_4, t_{999}\})$. In this chapter, we not only consider maximal convoys[18] but also valid (resp. closed) convoys [2]. Similar to swarm and closed swarm, a convoy becomes a valid convoy if it cannot be enlarged in terms of objects and timestamps.

Until now, we have considered that we have a group of objects that move close to each other for a long time interval. For instance, as shown in [14], moving clusters and different kinds of flocks virtually share essentially the same definition. Basically, the main difference is the use of clustering techniques. Flocks, for instance, usually consider a rigid definition of the radius while moving clusters and convoys apply a density-based clustering algorithm (e.g. DBScan[4]). 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[19]. Therefore, in the following, for brevity and clarity sake, we will mainly focus on convoy and density-based clustering algorithm.

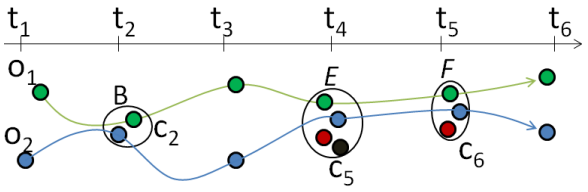


Figure 2.2: A group pattern example.

According to the previous definitions, the main difference between convoys and swarms is about the consecutiveness and non-consecutiveness of clusters during a time interval. In [32], Hwang et al. propose a general pattern, called a *group pattern*, which essentially is a combination of both convoy and closed swarm. 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 time point, 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 4. *Group Pattern[32]. Given a set of objects* O*, a minimum weight threshold* min_{wei}*, a set of disjointed convoys* $T_S = \{s_1, s_2, \ldots, s_n\}$*, a minimum number of convoys* min_c*.* (O, T_S) *is a group pattern if:*

$$\begin{cases} (1): (O, T_S) \textit{ is a closed swarm} min_o \textit{ w.r.t } min_c. \\ (2): \frac{\sum_{i=1}^{|T_S|} |s_i|}{|T_{db}|} \geq min_{wei}. \end{cases} \tag{2.4}$$

Note that min_c *is only applied for* T_S *(i.e.* $|T_S| \geq min_c$*).*

For instance, see Figure 2.2, with $min_t = 1$ and $min_o = 2$, we have a set of convoys $T_S = \{(\{o_1, o_2\}, \{t_1\}), (\{o_1, o_2\}, \{t_4, t_5\})\}$. Additionally, with $min_c = 1$, we have $(\{o_1, o_2\}, T_S)$ is a closed swarm of convoys because $|T_S| = 2 \geq min_c$, $|O| \geq min_o$ and (O, T_S) cannot be enlarged. Furthermore, with $min_{wei} = 0.5$, (O, T_S) is a group pattern since $\frac{|[t_1]|+|[t_4,t_5]|}{|T_{db}|} = \frac{3}{6} \geq min_{wei}$.

Previously, we overviewed patterns in which a group objects move together during some time intervals. However, mining patterns from individual object movement is also interesting. In [24], N. Mamoulis et al. propose the notion of *periodic pattern* 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 their work everyday. Informally, given an object's trajectory including $\mathcal{N}$ time-points, $\mathcal{T}_\mathcal{P}$ which is the number of timestamps that a pattern may re-appear. The object's trajectory is decomposed into $\lfloor \frac{\mathcal{N}}{\mathcal{T}_\mathcal{P}} \rfloor$ sub-trajectories. $\mathcal{T}_\mathcal{P}$ is data-dependent and has no definite value. For example, $\mathcal{T}_\mathcal{P}$ can be set to 'a day' in traffic control applications since many vehicles have daily patterns,

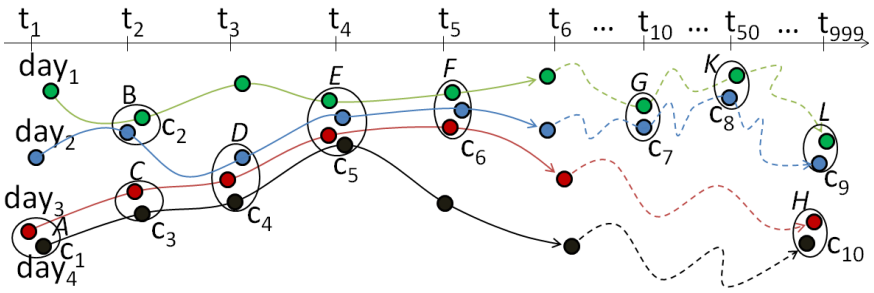


Figure 2.3: A periodic pattern example.

while annual animal migration patterns can be discovered by $\mathcal{T}_\mathcal{P}$ = 'a year'. For instance, see Figure 2.3, an object's trajectory is decomposed into daily sub-trajectories.

Essentially, a periodic pattern is a closed swarm discovered from $\lfloor \frac{\mathcal{N}}{\mathcal{T}_\mathcal{P}} \rfloor$ sub-trajectories. For instance, in Figure 2.3, we have 4 daily sub-trajectories and from them we extract potential periodic patterns such as $\{c_1, c_3, c_4, c_5\}$, $\{c_2, c_5, c_6\}$, etc. The main difference in periodic pattern mining is the preprocessing data step while the definition is similar to that of closed swarms. As we have provided the definition of closed swarms, we will mainly focus on closed swarm mining below.

2.2 Object Movement Pattern Mining

As mentioned before, many approaches have been proposed to extract movement patterns. Interested readers may refer to [14] where short descriptions of the most efficient or interesting patterns and approaches are proposed. For instance, Gudmundsson and van Kreveld [7], Vieira et al.[30] define a flock pattern, in which the same set of objects stay together in a circular region with a predefined radius, Kalnis et al.[19] propose the notion of *moving cluster*, while Jeung et al.[18] define a convoy pattern.

Jeung et al.[18] adopt the DBScan algorithm[4] to find candidate convoy patterns. The authors propose three algorithms 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. Another problem is related to the trajectory representation. Some trajectories may have missing timestamps or are measured at different time intervals. Therefore, the density measurements cannot be applied between trajectories with different timestamps. To address the problem of missing timestamps, 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 maximal pattern when it has at least k clusters during k consecutive timestamps. To accurate the discovery of convoys, H. Yoon et al. propose the notion of *valid convoy*[2] which can not be enlarged in terms of timestamps and objects.

Recently, Zhenhui Li et al.[23] propose the concept of swarm and closed swarm and the *ObjectGrowth* algorithm to extract closed swarms. The ObjectGrowth method is a depth-first-search framework based on the objectset search space (i.e., the collection of all subsets of O_{db}). For the search space of O_{db}, they perform depth-first search of all subsets of O_{db} through a pre-order tree traversal. Even though, the search space remains still huge for enumerating the objectsets in O $(2^{|O_{db}|})$. To speed up the search process, they propose two pruning rules. The first pruning rule, called *Apriori Pruning*, is used to stop traversal the subtree when we find further traversal that cannot satisfy min_t. The second pruning rule, called *Backward Pruning*, makes use of the closure property. It checks whether there is a superset of the current objectset, which has the same maximal corresponding timeset as that of the current one. If so, the traversal of the subtree under the current objectset is meaningless. After pruning the invalid candidates, the remaining ones may or may not be closed swarms. Then a *Forward Closure Checking* is used to determine whether a pattern is a closed swarm or not.

In [32], 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 [1] to mine valid group patterns. The latter is based on idea similar to the FP-growth algorithm [15].

To discover group of moving objects on streaming trajectories, L.-A. Tang et al. [28] present the concept of traveling companions. In order to efficiently extract traveling companions, the authors first propose the models of closed companion candidates and smart intersection to accelerate data processing. Then, a data structure termed traveling buddy is designed to facilitate scalable and flexible companion discovery from streaming trajectories. The traveling buddies are micro-groups of objects that are tightly bound together. By only storing the object relationships rather than their spatial coordinates, the buddies can be dynamically maintained along trajectory stream with low cost. Based on traveling buddies, the system can discover companions without accessing the object details.

Patterns that are mined from trajectories are called trajectory patterns and characterize interesting behaviors of single object or group of moving objects. Giannotti et al. [65] presented an algorithm to find frequent movement patterns that represent cumulative behavior of moving objects where a pattern, called *T-pattern*, was defined as a sequence of points with temporal transitions between consecutive points. A *T-pattern* is discovered if its spatial and temporal components approximately correspond to the input sequences (trajectories). The meaning of these patterns is that different objects visit the same places with similar time intervals. Once the patterns are discovered, the classical sequence mining algorithms can be applied to find frequent patterns. Crucial to the determination of *T-patterns* is the definition of the visiting regions. For this, the Region-of-Interest (RoI) notion was proposed. A RoI is defined as a place visited by many objects. Additionally, the duration of stay can be taken into account. The idea behind RoI is to divide the working region into cells and count the number of trajectories that intersect the cell. The algorithm

Table 2.2: Overview of movement pattern methods and applications.

Problem	Application	Based-Method	Selected literature
Trajectory clustering, Trajectory aggregation, Trajectory generalization	Cars, evacuation traces ,landings and interdictions of migrant boats	OPTICS	Rinzivillo et al. [59] Andrienko et al. [60] [61] [62] Lee et al. [22]
Moving clusters	Migrating animals, flocks, convoys of vehicles, swarms, gradual trajectories	DBSCAN Gridding, DBSCAN	Kalnis et al. [19] Jeung et al. [18] Vieira et al. [30] Li et al. [23] Aung et al. [2] Phan et al. [9] [10] [12] Tang et al. [28]
Extracts important places from trajectories	People's trajectories	DBSCAN, Incremental clustering	Palma et al. [63] Kang et al. [64]
Trajectory patterns	Fleet of trucks	Density of spatial regions	Giannotti et al. [65]
Representative trajectories	Migrating animals	DBSCAN	Phan et al. [11]

for finding popular regions was proposed, which accepted the grid with cell densities and a density threshold d as input. The algorithm scans the cells and tries to expand the region in four directions (left, right, up, down). The direction that maximizes the average cell density is selected and the cells are merged. After the regions of interest are obtained, the sequences can be created by following every trajectory and matching the regions of interest they intersect. The timestamps are assigned to the regions in two ways: (1) Using the time when the trajectory entered the region or (2) Using the starting time if the trajectory started in that region. Consequently, the sequences are used in mining frequent *T-patterns*.

As we can recognize that there are many kinds of patterns and algorithms, i.e. Table 2.2 summarizes the categories of movement patterns, thus it is costly and time consuming to mine and manage all them. Additionally, in some real world applications, i.e. cars, the new object information is continuously reported and thus it demands an innovative solution to manage them. These challenging issues are addressed in the next Chapter, All in One: Mining Multiple Movement Patterns.

CHAPTER

3 All in One: Mining Multiple Movement Patterns

Preamble

Due to the emergence of many different kinds of object movement patterns in recent years, different approaches have been proposed to extract them. However, each approach only focuses on mining a specific kind of patterns. In addition to being a painstaking task due to the large number of algorithms used to mine and manage patterns, it is also time consuming. Moreover, we have to execute these algorithms again whenever new data are added to the existing database. To address these issues, we first redefine movement patterns in the itemset context. Secondly, we propose a unifying approach, named GeT_Move, which uses a frequent closed itemset-based object movement pattern-mining algorithm to mine and manage different patterns. GeT_Move is developed in two versions which are GeT_Move and Incremental GeT_Move. To optimize the efficiency and to free the parameters setting, we further propose a Parameter Free Incremental GeT_Move algorithm. Comprehensive experiments are performed on real and large synthetic datasets to demonstrate the effectiveness and efficiency of our approaches.

3.1 Object Movement Patterns in Itemset Context

For clarity sake, we remind some notions introduced in Chapter 2. First of all, let us assume that we have a group of moving objects $O_{db} = \{o_1, o_2, \ldots, o_z\}$, a set of timestamps $T_{db} = \{t_1, t_2, \ldots, t_n\}$. Then $O = \{o_{i_1}, o_{i_2}, \ldots, o_{i_p}\} (O \subseteq O_{db})$ 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 min_o be the user-defined threshold standing for a minimum number of objects and

Table 3.1: Cluster matrix from our running example in Figure 1.1.

T_{db}		t_1	t_2		t_3	t_4	t_5	t_{10}	t_{50}	t_{999}	
Clusters C_{db}		c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
O_{db}	o_1		1			1	1	1	1	1	
	o_2		1		1	1	1	1	1	1	
	o_3	1		1	1	1	1				1
	o_4	1		1	1	1					1

min_t the minimum number of timestamps. Thus $|O|$ (resp. $|T|$) must be greater than or equal to min_o (resp. min_t).

Database of clusters. A database of clusters, $C_{db} = \{C_{t_1}, C_{t_2}, \ldots, C_{t_n}\}$, is the collection of snapshots of the moving object clusters at timestamps $\{t_1, t_2, \ldots, t_n\}$. Note that an object could belong to several clusters at one timestamp (i.e. cluster overlapping). Given a cluster $c \in C_t (\in C_{db})$ and $c \subseteq O_{db}$, $|c|$ and $t(c)$ are respectively used to denote the number of objects belong to cluster c and the timestamp that c involved in. To make our framework more general, we take clustering as a preprocessing step.

Basically, patterns are evolution of clusters over time. Therefore, to manage the evolution of clusters, we need to analyze 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 can have a set of items (called itemset). The key issue 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 a pattern. To describe cluster evolution, moving object data is presented as a cluster matrix from which patterns can be extracted.

Definition 5. *Cluster Matrix. Assume that we have a set of clusters* C_{db}. *A cluster matrix is thus a matrix of size* $|O_{db}| \times |C_{db}|$ *such that 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 by applying clustering techniques.*

For instance, the data from our illustrative example (Figure 1.1) is presented in a cluster matrix in Table 3.1. Object o_3 belongs to the cluster c_1 at timestamp t_1 and thus the matrix cell $(o_3\text{-}c_1)$ is 1, meanwhile the matrix cell $(o_1\text{-}c_1)$ is empty because object o_1 does not belong to cluster c_1.

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 $\Upsilon = \{c_1, c_2, \ldots, c_p\}$ with life time $T_\Upsilon = \{t(c_1), t(c_2), \ldots, t(c_p)\}$ where $t(c_1) < t(c_2) < \ldots < t(c_p)$, $\forall i : t(c_i) \in T_{db}, c_i \in C_{db}$. The support of the itemset Υ, denoted $\sigma(\Upsilon)$, is the number of common objects in

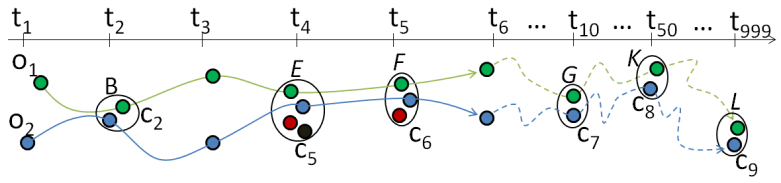


Figure 3.1: A swarm from our running example in Figure 1.1.

every items belonging to Υ, $O(\Upsilon) = \bigcap_{i=1}^{p} c_i$. Additionally, the length of Υ, denoted $|\Upsilon|$, is the number of items or timestamps ($= |\Upsilon| = |T_\Upsilon|$).

For instance, in Table 3.1, for a support value of 2 we have: $\Upsilon = \{c_2, c_7\}$ veryfying $\sigma(\Upsilon) = 2$. All the items (resp. clusters) of Υ, c_1 and c_7, 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 an 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 2.1 from frequent itemsets as follows:

Property 1. *Swarm. Given a frequent itemset* $\Upsilon = \{c_1, c_2, \ldots, c_p\}$. $(O(\Upsilon), T_\Upsilon)$ *is a swarm if, and only if:*

$$\begin{cases} (1) : \sigma(\Upsilon) \geq min_o \\ (2) : |\Upsilon| \geq min_t \end{cases} \tag{3.1}$$

Proof. After construction, we have $\sigma(\Upsilon) \geq min_o$ and $\sigma(\Upsilon) = |O(\Upsilon)|$ then $|O(\Upsilon)| \geq min_o$. Additionally, as $|\Upsilon| \geq min_t$ and $|\Upsilon| = |T_\Upsilon|$ then $|T_\Upsilon| \geq min_t$. Furthermore, $\forall t_j \in T_\Upsilon, O(\Upsilon) \subseteq c_j$, means that at any timestamps, 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 3.1, for the frequent itemset $\Upsilon = \{c_2, c_5, c_6, c_7, c_8, c_9\}$ we have $(O(\Upsilon) = \{o_1, o_2\}, T_\Upsilon = \{t_2, t_4, t_5, t_{10}, t_{50}, t_{999}\})$ which is a swarm with support threshold $min_o = 2$ and $min_t = 2$. We can notice that $\sigma(\Upsilon) = 2 \geq min_o$ and $|\Upsilon| = 6 \geq min_t$.

Essentially, a closed swarm is a swarm which satisfies the *object-closed* and *time-closed* conditions therefore closed-swarm property is as follows:

Property 2. *Closed Swarm. Given a frequent itemset* $\Upsilon = \{c_1, c_2, \ldots, c_p\}$. $(O(\Upsilon), T_\Upsilon)$ *is a closed swarm if and only if:*

$$\begin{cases} (1) : (O(\Upsilon), T_\Upsilon) \textit{ is a swarm.} \\ (2) : \nexists \Upsilon' \textit{ s.t } O(\Upsilon) \subset O(\Upsilon'), T_{\Upsilon'} = T_\Upsilon \textit{ and } (O(\Upsilon'), T_\Upsilon) \textit{ is a swarm.} \\ (3) : \nexists \Upsilon' \textit{ s.t. } O(\Upsilon') = O(\Upsilon), T_\Upsilon \subset T_{\Upsilon'} \textit{ and } (O(\Upsilon), T_{\Upsilon'}) \textit{ is a swarm.} \end{cases} \tag{3.2}$$

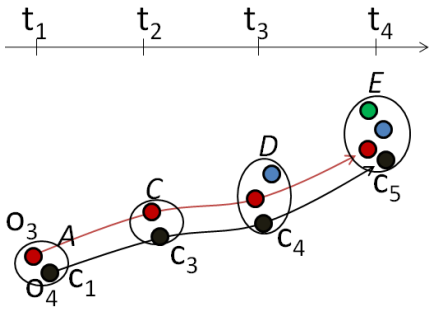


Figure 3.2: A convoy from our running example in Figure 1.1.

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 6*. □

According to the *Definition 3*, a convoy is a swarm which satisfies the lifetime consecutiveness condition. Therefore, for an itemset, we can extract a convoy if the following property holds:

Property 3. *Convoy. Given a frequent itemset* $\Upsilon = \{c_1, c_2, \ldots, c_p\}$. $(O(\Upsilon), T_\Upsilon)$ *is a convoy if and only if:*

$$\begin{cases} (1) : (O(\Upsilon), T_\Upsilon) \text{ is a swarm.} \\ (2) : \forall j, 1 \leq j < p : t(c_{j+1}) = t(c_j) + 1. \end{cases} \tag{3.3}$$

Proof. After construction, we obtain $(O(\Upsilon), T_\Upsilon)$ which is a swarm. Additionally, if Υ satisfies the condition (2), it means that the Υ's lifetime is consecutive. Consequently, $(O(\Upsilon), T_\Upsilon)$ is a convoy according to the *Definition 3*. □

For instance, see Table 3.1 and Figure 3.2, for the frequent itemset $\Upsilon = \{c_1, c_3, c_4, c_5\}$ we have $(O(\Upsilon) = \{o_3, o_4\}, T_\Upsilon = \{t_1, t_2, t_3, t_4\})$ is a convoy with support threshold $\min_o = 2$ and $\min_t = 2$.

Please remember that a group pattern is a set of disjointed convoys which share the same objects, but in different time intervals. Therefore, the group pattern property is as follows:

Property 4. *Group Pattern. Given a frequent itemset* $\Upsilon = \{c_1, c_2, \ldots, c_p\}$, *a mininum weight* $\min_{wei}$, *a minimum number of convoys* $\min_c$, *a set of consecutive time segments*

Table 3.2: Periodic cluster matrix in Figure 2.3.

T_{db}		t_1	t_2		t_3	t_4	t_5	t_{10}	t_{50}	t_{999}	
Clusters C_{db}		c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
ST_{db}	st_1		1			1	1	1	1	1	
	st_2		1		1	1	1	1	1	1	
	st_3	1		1	1	1	1				1
	st_4	1		1	1	1					1

$T_S = \{s_1, s_2, \ldots, s_n\}$. $(O(\Upsilon), T_S)$ *is a group pattern if and only if:*

$$\begin{cases} (1): |T_S| \geq min_c. \\ (2): \forall s_i, s_i \subseteq T_\Upsilon, |s_i| \geq min_t. \\ (3): \bigcap_{i=1}^{n} s_i = \emptyset, \bigcap_{i=1}^{n} O(s_i) = O(\Upsilon). \\ (4): \forall s \notin T_S, s \textit{ is a convoy}, O(\Upsilon) \not\subseteq O(s). \\ (5): \frac{\sum_{i=1}^{n} |s_i|}{|T|} \geq min_{wei}. \end{cases} \tag{3.4}$$

Proof. If $|T_S| \geq min_c$ then we know that at least min_c consecutive time intervals s_i in T_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 3*). Now, T_S actually is a set of convoys of $O(\Upsilon)$ and if $\bigcap_{i=1}^{n} s_i = \emptyset$ then T_S is a set of disjointed convoys. A little bit further, if $\forall s \notin T_S, s$ is a convoy and $O(\Upsilon) \not\subseteq O(s)$ then $\nexists T_{S'}$ s.t. $T_S \subset T_{S'}$ and $\bigcap_{i=1}^{|T_{S'}|} O(s_i) = O(\Upsilon)$. Therefore, $(O(\Upsilon), T_S)$ cannot be enlarged in terms of *number of convoys*. Similarly, if $\bigcap_{i=1}^{n} O(s_i) = O(\Upsilon)$ then $(O(\Upsilon), T_S)$ cannot be enlarged in terms of *objects*. Consequently, $(O(\Upsilon), T_S)$ is a closed swarm of disjointed convoys because $|O(\Upsilon)| \geq min_o, |T_S| \geq min_c$ and $(O(\Upsilon), T_S)$ cannot be enlarged (*Definition 6*). Finally, if $(O(\Upsilon), T_S)$ satisfies condition (5) then it is a valid group pattern due to *Definition 4*. □

As mentioned before, the main difference in periodic pattern mining is the input data while the definition is similar to that of closed swarm. The cluster matrix which is used for periodic mining can be defined as follows:

Definition 6. *Periodic Cluster Matrix (PCM). Periodic cluster matrix is a cluster matrix with some differences as follows: 1) Each object* o *is a sub-trajectory* st*, 2)* ST_{db} *is a set of all sub-trajectories in dataset.*

For instance, see Table 3.2, an object's trajectory is decomposed into 4 sub-trajectories and from them a periodic cluster matrix can be generated by applying clustering techniques. Assume that we can extract a frequent itemset $\Upsilon = \{c_1, c_2, \ldots, c_p\}$ from periodic cluster matrix, the periodic pattern can be defined as follows:

Property 5. *Periodic Pattern. Given a mininum weight* $\min_{wei}$*, a frequent itemset* $\Upsilon = \{c_1, c_2, \ldots, c_p\}$ *which is extracted from periodic cluster matrix.* $(ST(\Upsilon), (T)_\Upsilon)$ *is a periodic pattern if and only if* $(ST(\Upsilon), (T)_\Upsilon)$ *is a closed swarm. Note that* $ST(\Upsilon) = \bigcap_{i=1}^{p} c_i$

Above, we presented some useful properties to extract object movement 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 movement patterns. We thus start the proof process by analyzing the swarm extracting issue. 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* $\min sup = \min_o$*. All swarms* (O, T) *can be extracted from* FI.

Proof. Let us assume that (O, T) is a swarm. Note, $T = \{t_1, t_2, \ldots, t_m\}$. According to the *Definition 1* we know that $|O| \geq \min_o$. If (O, T) is a swarm then $\forall t_i \in T, \exists c_{t_i}$ s.t. $O \subseteq c_{t_i}$ with $t(c_{t_i}) = t_i$ therefore $\bigcap_{i=1}^{m} c_{t_i} = O$. Additionally, we know that $\forall c_{t_i}$, c_{t_i} is an item so $\exists \Upsilon = \bigcup_{i=1}^{m} c_{t_i}$ is an itemset and $O(\Upsilon) = \bigcap_{i=1}^{m} c_{t_i} = O, T_\Upsilon = \bigcup_{i=1}^{m} t_i = T$. Therefore, $(O(\Upsilon), T_\Upsilon)$ is a swarm. So, (O, T) is extracted from Υ. Furthermore, $\sigma(\Upsilon) = |O(\Upsilon)| = |O| \geq \min_o$ then Υ is a frequent itemset and $\Upsilon \in FI$. Finally, $\forall (O, T)$ 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 $\forall$ 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, if $Swarm, CSwarm, Convoy, MCluster$ respectively contain all swarms, closed-swarms, convoys and moving clusters then we have: $CSwarm \subseteq Swarm$, $Convoy \subseteq Swarm$ and $MCluster \subseteq Swarm$. By applying *Lemma 5*, we retrieve all swarms from frequent itemsets. Since, a set of closed swarms, a set of convoys and a set of moving clusters are subsets of swarms and they can therefore be completely extracted from frequent itemsets. Additionally, all periodic patterns also can be extracted because they are similar to closed swarms. Now, we will consider group patterns and we show that all of them can be directly extracted from the set of all frequent itemsets.

Lemma 2. *Given* $FI = \{\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_l\}$ *contains all frequent itemsets mined from cluster matrix with* $\min sup = \min_o$*. All group patterns* (O, T_S) *can be extracted from* FI.

Proof. $\forall (O, T_S)$ is a valid group pattern, we have $\exists T_S = \{s_1, s_2, \ldots, s_n\}$ and T_S is a set of disjointed convoys of O. Therefore, (O, T_{s_i}) is a convoy and $\forall s_i \in T_S, \forall t \in T_{s_i}, \exists c_t$ 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_S) is a valid group pattern; therefore, $|O| \geq \min_o$ so $|O(\Upsilon)| \geq \min_o$. Consequently, Υ is a frequent itemset and $\Upsilon \in FI$ because Υ is an itemset and $\sigma(\Upsilon) = |O(\Upsilon)| \geq \min_o$. Consequently, $\forall (O, T_S), \exists \Upsilon \in FI$ s.t.

(O, T_S) can be extracted from Υ and therefore all group patterns can be extracted from FI. □

As we have shown that patterns such as swarms, closed swarms, convoys, group patterns can be similarly mapped into frequent itemset context. However, mining all frequent itemsets is cost prohibitive in some cases. Fortunately, the set of frequent closed itemsets has been proved to be a condensed collection of frequent itemsets, i.e., both a concise and lossless representation of a collection of frequent itemsets [58]. They are concise since a collection of frequent closed itemsets is orders of magnitude smaller than the corresponding collection of frequents. This allows the use of very low minimum support thresholds. Moreover, they are lossless, because it is possible to derive the identity and the support of every frequent itemsets in the collection from them. Therefore, we only need to extract frequent closed itemsets (FCIs) and then to scan them with properties to obtain the corresponding object movement patterns instead of having to mine all frequent itemsets (FIs).

3.2 Frequent Closed Itemset-based Object Movement Pattern Mining Algorithm

Recently, patterns have been redefined in the itemset context. 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 3.3.

In the first step, a clustering approach (Figure 3.3-(1)) is applied at each time-stamp to group objects into different clusters. For each timestamp t, we have a set of clusters C_t. Moving object data can thus be converted to a cluster matrix CM (Table 3.1).

3.2.1 GeT_Move

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

In this chapter, LCM algorithm[38] is applied to extract FCIs as it is known to be a very efficient algorithm. The key feature of the LCM algorithm is that after discovering a FCI X, it generates a new generator $X[i]$ by extending X with a frequent item $i, i \notin X$. Using a total order relation on frequent items, LCM verifies if $X[i]$ violates this order by performing tests using only the *tidset*[1] of X, called $\mathcal{T}(X)$, and those of the frequent items i. If $X[i]$ is not discarded, then $X[i]$ is an order preserving generator of a new FCI. Then, its closure is computed using the previously mentioned tidsets.

In this process, we discard some useless itemset candidates. In object movement patterns, items (resp. clusters) must belong to different timestamps and therefore items (resp.

1. Called *tidlists* in [38].

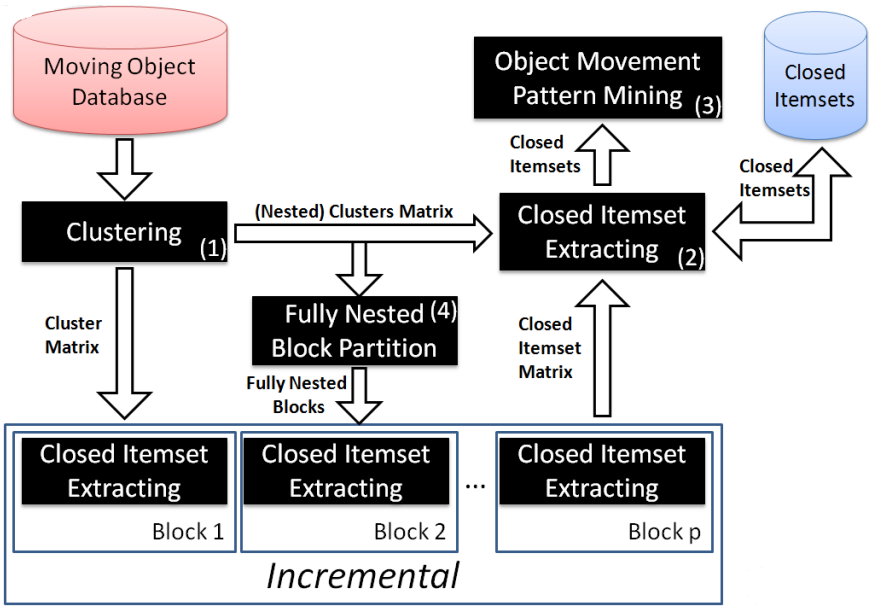


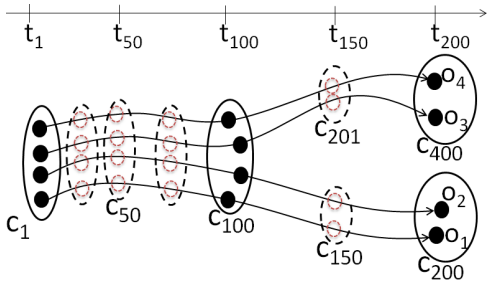
Figure 3.3: The main process.

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.

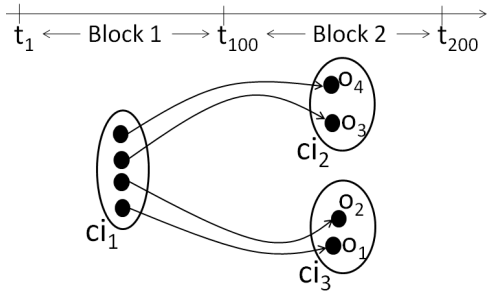
Thanks to the above characteristic, we now have the maximum length of the FCIs which is the number of timestamps $|T_{db}|$. Additionally, the LCM search space only depends on the number of objects (transactions) $|O_{db}|$ and the maximum length of itemsets $|T_{db}|$. Consequently, by using LCM and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

The pseudo code of *GeT_Move* is described in *Algorithm 5*. The core of *GeT_Move* algorithm is based on the LCM algorithm which has been slightly modified by adding the pruning rule and by extracting patterns from FCIs. The initial value of FCI X is empty and then we start by putting item i into X (lines 2-3). By adding i into X, we have X[i] and if X[i] is a FCI then X[i] is used as a generator of a new FCI, call LCM_Iter$(X, \mathcal{T}(X), i(X))$ (lines 4-5). In LCM_Iter, we first check properties presented in Section 3.1 (line 8) for FCI X. Next, for each transaction $t \in \mathcal{T}(X)$, we add all items j, which are larger than $i(X)$ and satisfy the pruning rule, into occurrence set $\mathcal{J}[j]$ (lines 9-11). Next, for each $j \in \mathcal{J}[j]$, we check to see if $\mathcal{J}[j]$ is a FCI, and if so, then we recall LCM_Iter with the new generator (lines 12-14). After terminating the call for $\mathcal{J}[j]$, the memory for $\mathcal{J}[j]$ is released for the future use in $\mathcal{J}[k]$ for $k < j$ (lines 15).

Regarding to the PatternMining sub-function (lines 16-37), the algorithm basically checks properties of the itemset X to extract patterns. If X satisfies the min_t condition



(a) The entire dataset.



(b) Data after applying frequent closed itemsets mining on Blocks.

Figure 3.4: A case study example. (b)-ci_1 (resp. ci_2, ci_3) is a frequent closed itemset extracted from block 1 (resp. block 2).

then X is a closed swarm (lines 18-19). After that, we check the consecutive time constraint for convoy and moving cluster (lines 21-22) and then if the convoy satisfies min_t condition and correctness in terms of object containing (line 31), outputs convoy (line 32). Next, we put the convoy into a group pattern $gPattern$ (line 33) and then output $gPattern$ if it satisfies the min_c condition and min_{wei} condition at the end of scanning X (line 37). Regarding to the moving cluster mc, we check the integrity at each pair of consecutive timestamps (line 24). If mc satisfies the condition then the previous item x_k will be merged into mc (line 25). If not, we check the min_t condition for $mc \cup x_k$ and if it is satisfied then we output $mc \cup x_k$ as a moving cluster.

3.2.2 Incremental GeT_Move

Usually, the transaction length can be large corresponding to $|T_{db}|$. Furthermore, FCI mining approaches can be slowed down when working with long transactions. Thus, 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 this issue, we propose the *Incremental GeT_Move* algorithm. The basic idea is to shorten the transactions by splitting the trajectories (resp. cluster matrix CM) into

Algorithm 1: GeT_Move

Input : Occurrence sets $\mathcal{J}$, int min_o, int min_t, set of items C_{db}, double θ, int min_c, double min_{wei}

begin
  $X := I(\mathcal{T}(\emptyset))$; //The root
  for $i := 1$ *to* $|C_{db}|$ **do**
    if $|\mathcal{T}(X[i])| \geq min_o$ *and* $|X[i]|$ *is closed* **then**
      LCM_Iter($X[i], \mathcal{T}(X[i]), i$);
LCM_Iter($X, \mathcal{T}(X), i(X)$)
begin
  PatternMining(X, min_t); /*X is a pattern?*/
  foreach *transaction* $t \in \mathcal{T}(X)$ **do**
    foreach $j \in t, j > i(X), j.time \notin time(X)$ **do**
      insert j to $\mathcal{J}[j]$;
  foreach $j \in \mathcal{J}[j]$ *in the decreasing order* **do**
    if $|\mathcal{T}(\mathcal{J}[j])| \geq min_o$ *and* $\mathcal{J}[j]$ *is closed* **then**
      LCM_Iter($\mathcal{J}[j], \mathcal{T}(\mathcal{J}[j]), j$);
    Delete $\mathcal{J}[j]$;
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* $\frac{size(gPattern)}{|T_{db}|} \geq min_{wei}$ **then**
      output gPattern; /*Group Pattern*/

Where: X is itemset, $X[i] := X \cup i$, $i(X)$ is the last item of X, $\mathcal{T}(X)$ is list of tractions that X belongs to, $\mathcal{J}[j] := \mathcal{T}(X[j])$, $j.time$ is time index of item j, $time(X)$ is a set of time indexes of X, $|\mathcal{T}(convoy)|$ is the number of transactions that the $convoy$ belongs to, $|gPattern|$ and $size(gPattern)$ respectively are the number of convoys and the total length of the convoys in $gPattern$.

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 3.4, 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 (insteads of 200). Additionally,

Table 3.3: Closed Itemset Matrix.

Block B		b_1	b_2	
Frequent Closed Itemsets CI		ci_1	ci_2	ci_3
O_{db}	o_1	1		1
	o_2	1		1
	o_3	1	1	
	o_4	1	1	

the original data can be greatly compressed (i.e. Figure 3.4b) and only 3 items remain: ci_1, ci_2, ci_3. Consequently, the process is much improved.

Definition 7. *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 $B = \{b_1, b_2, \ldots, b_p\}$ with $|T_{b_1}| = |T_{b_2}| = \ldots = |T_{b_p}|, \bigcup_{i=1}^{p} b_i = CM$ and $\bigcap_{i=1}^{p} b_i = \emptyset$. Given a set of FCI collections $CI = \{CI_1, CI_2, \ldots, CI_p\}$ 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 8. *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 frequent closed itemset* ci.

For instance, see Table 3.3, we have two sets of FCIs $CI_1 = \{ci_1\}, CI_2 = \{ci_2, ci_3\}$ which are respectively extracted from blocks b_1, b_2. Closed itemset matrix $CIM = CI_1 \cup CI_2$ means that CIM is created by horizontally connecting CI_1 and CI_2. Consequently, we have CIM as in Table 3.3.

We have already provided *blocks* to compress original data. Now, by applying FCI mining on the closed itemset matrix CIM, we are able to retrieve all FCIs from corresponding data. Note that items (in CIM) which are in the same block cannot be in the same frequent closed itemset.

Lemma 3. *Given a cluster matrix* CM *which is vertically split into a set of blocks* $B = \{b_1, b_2, \ldots, b_p\}$ *so that* $\forall \Upsilon, \Upsilon$ *is a frequent closed itemset and* Υ *is extracted from* CM *then* Υ *can be extracted from 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 $\Upsilon = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$ 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 $|O(\Upsilon)| \geq min_o$; therefore, $|O(\gamma_i)| \geq min_o$

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 $O(\Upsilon) = O(\gamma_1) \cap O(\gamma_2) \cap \ldots \cap O(\gamma_i) \cap \ldots \cap O(\gamma_p)$ then we have: $\exists \Upsilon'$ s.t. $O(\Upsilon') = O(\gamma_1) \cap O(\gamma_2) \cap \ldots \cap O(\Psi) \cap \ldots \cap O(\gamma_p)$. 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 $\forall \Upsilon, \Upsilon = \{\gamma_1, \gamma_2, \ldots, \gamma_p\}$ is a FCI extracted from CM, $\forall \gamma_i \in \Upsilon$, γ_i must 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 Pattern 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 difference between the code of *Incremental GeT_Move* and *GeT_Move* is the *Update* subfunction. In this function, we, step by step, generate the closed itemset matrix from blocks (line 14 and lines 22-26). Next, we apply *GeT_Move* to extract patterns (line 5).

3.3 Preliminarily 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 implementations of our proposed algorithms are also integrated in our demonstration system[8] and it is public online [2]. As in [23] [33], the two following datasets [3] 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.

Similar to [23] [18], we first use linear interpolation to fill in the missing data. For study purposes, we needed the objects to stay together for at least min_t timestamps. As in [23] [18], DBScan[4] ($MinPts = 2, Eps = 0.001$) is applied to generate clusters at each timestamp.

2. www.lirmm.fr/~phan/index.jsp
3. http://www.movebank.org

Algorithm 2: Incremental GeT_Move

Input : Occurrence sets K, int min_o, int min_t, double θ, set of Occurrence sets (blocks) B, int min_c, double min_{wei}

```
1  begin
2      K := ∅; CI := ϕ; int item_total := 0;
3      foreach b ∈ B do
4          LCM(b, min_o, I_b);
5      GeT_Move(K, min_o, min_t, CI, θ, min_c, min_wei);
6  LCM(Occurrence sets 𝒥, int σ_0, set of items C)
7  begin
8      X := I(𝒯(∅)); //The root
9      for i := 1 to |C| do
10         if |𝒯(X[i])| ≥ min_o and |X[i]| is closed then
11             LCM_Iter(X[i], 𝒯(X[i]), i);
12 LCM_Iter(X, 𝒯(X), i(X))
13 begin
14     Update(K, X, 𝒯(X), item_total++);
15     foreach transaction t ∈ 𝒯(X) do
16         foreach j ∈ t, j > i(X), j.time ∉ time(X) do
17             insert j to 𝒥[j];
18     foreach j, 𝒥[j] ≠ ϕ in the decreasing order do
19         if |𝒯(𝒥[j])| ≥ min_o and 𝒥[j] is closed then
20             LCM_Iter(𝒥[j], 𝒯(𝒥[j]), j);
21         Delete 𝒥[j];
22 Update(K, X, 𝒯(X), item_total)
23 begin
24     foreach t ∈ 𝒯(X) do
25         insert item_total into K[t];
26     CI := CI ∪ item_total;
```

3.3.1 Effectiveness

We proved that mining object movement patterns can be similarly mapped into itemset mining issue. Therefore, in theoretical way, our approaches can provide the correct results. Experimentally, we do a further comparison, we first obtain the object movement patterns by employing traditional algorithms such as, CMC, CuTS* [4] (convoy mining), ObjectGrowth (closed swarm mining) as well as our approaches. To apply our algorithms, we split the cluster matrix into blocks such as each block b contains 25 consecutive

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

(a) One of discovered closed swarms.

(b) One of discovered convoys.

(c) One of discovered group patterns.

Figure 3.5: An example of patterns discovered from Swainsoni dataset.

timestamps. Additionally, to retrieve all the patterns, in the reported experiments, the default value of min_o 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.

The results show that our proposed approaches obtain the same results compared to the traditional algorithms. An example of patterns is illustrated in Figure 3.5. For instance, see Figure 3.5a, a closed swarm is discovered within a frequent closed itemset. We can consider that the three Swainsonies move together from North America to South America and some time they leave each other, e.g. at the beginning of Mexico. Furthermore, from the itemset, a convoy and a group pattern are also extracted (i.e. Figures 3.5b, 3.5c). In the convoy pattern, the three object move together consecutively from Guatemala to Argentina meanwhile in group pattern, there are two consecutive segments: 1) from USA to Mexico, 2) from Guatemala to Argentina.

3.3.2 Efficiency

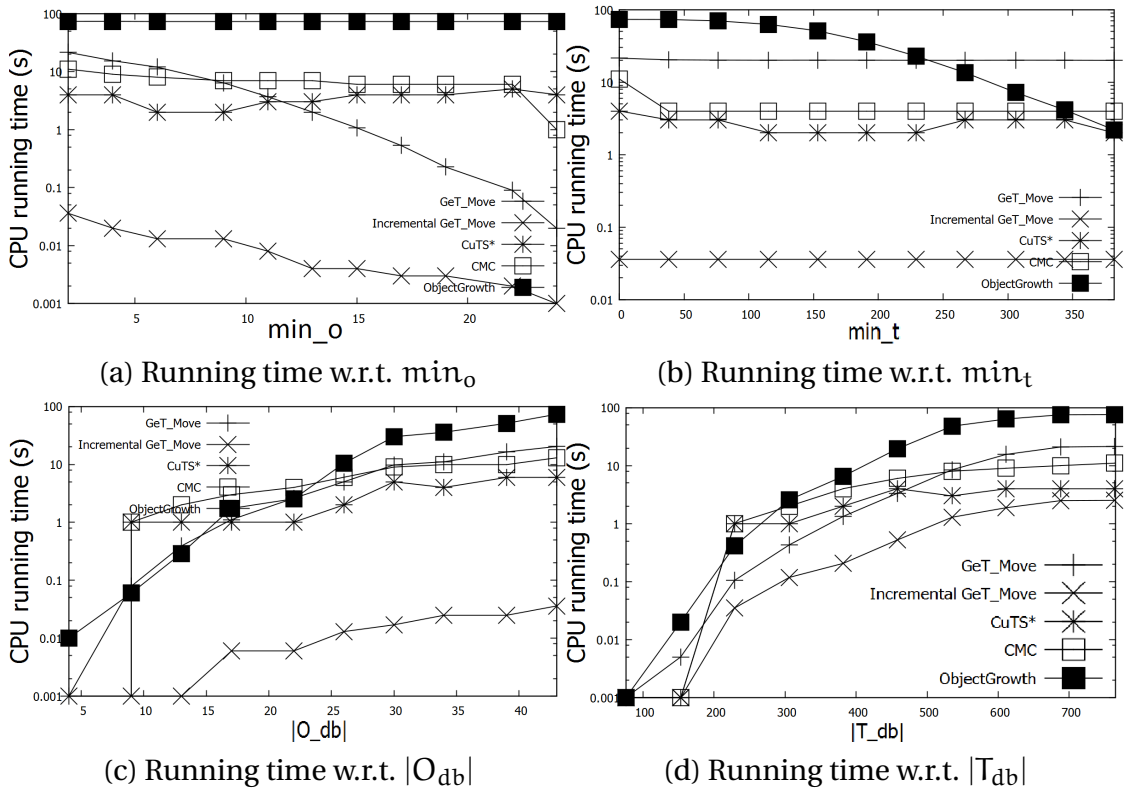
Incremental GeT_Move and GeT_Move

To show the efficiency of our algorithms, we generate larger synthetic datasets using Brinkhoff's network[5]-based generator of moving objects as in [23]. We generate 500 objects ($|O_{db}| = 500$) for 10^4 timestamps ($|T_{db}| = 10^4$) using the generator's default map with low moving speed (250). There are 5×10^6 points in total. DBScan ($MinPts = 3, Eps = 300$) is applied to obtain clusters for each timestamp.

In the efficiency comparison, we employ $CMC, CuTS^*$ and $ObjectGrowth$. Note that, in [23], $ObjectGrowth$ outperforms VG-Growth[32] (a group patterns mining algorithm) in terms of performance and therefore we will only consider $ObjectGrowth$ and not both. Note that GeT_Move and Incremental GeT_Move extracted closed swarms, convoys and group patterns meanwhile $CMC, CuTS^*$ only extracted convoys and $ObjectGrowth$ extracted closed swarms.

Efficiency w.r.t. min_o, min_t. Figures 3.6a, 3.7a, 3.8a show running time w.r.t. min_o. 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 min_o and therefore the change of min_o 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 long transactions. Meanwhile, thanks to blocks, the number of items is greatly reduced and transactions are not long as the ones in GeT_Move.

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



(a) Running time w.r.t. $\min_o$

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

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

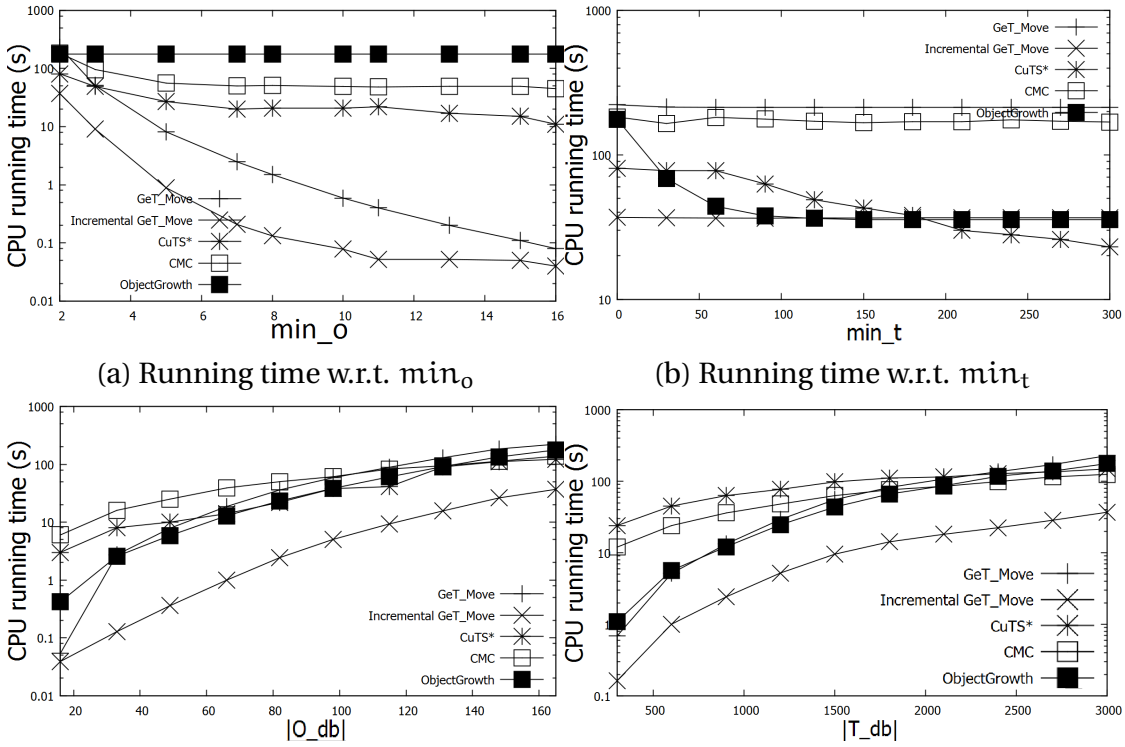
(d) Running time w.r.t. $|T_{db}|$

Figure 3.6: Running time on Swainsoni dataset.

Figures 3.6b, 3.7b, 3.8b show running time w.r.t. $\min_t$. In almost all cases, our approaches outperform other algorithms. See Figures 3.7b, 3.8b, with low $\min_t$, our algorithm is much faster than the others. However, when $\min_t$ is higher ($\min_t > 200$ in Figure 3.7b, $\min_t > 20$ in Figure 3.8b) our algorithms take more time than CuTS* and ObjectGrowth. This is because with high value of $\min_t$, the number of patterns is significantly reduced, i.e. Figures 3.9b, 3.10b, 3.11b, and therefore CuTS* and ObjectGrowth is faster. Meanwhile, GeT_Move and Incremental GeT_Move have to work with FCIs.

Efficiency w.r.t. $|O_{db}|,|T_{db}|$. Figures 3.6c-d, Figures 3.7c-d, Figures 3.8c-d show the running time when varying $|O_{db}|$ and $|T_{db}|$ respectively. In all figures, Incremental GeT_Move outperforms other algorithms. However, with synthetic data (Figure 3.8d) and lowest values of $\min_o = 2$ and $\min_t = 1$, GeT_Move is a little bit faster than Incremental GeT_Move. This is the clue to the fact that Incremental GeT_Move does not have any information to obtain the better partitions (blocks).

Scalability w.r.t. $\min_o$. We can consider that the running time of algorithms does not change significantly when varied $\min_t, |O_{db}|, |T_{db}|$ in synthetic data (Figures 3.8). However, they are quite different when varying $\min_o$ (default $\min_t = 1$). Therefore, we generate another large synthetic data to test the scalability of algorithms on $\min_o$. The dataset



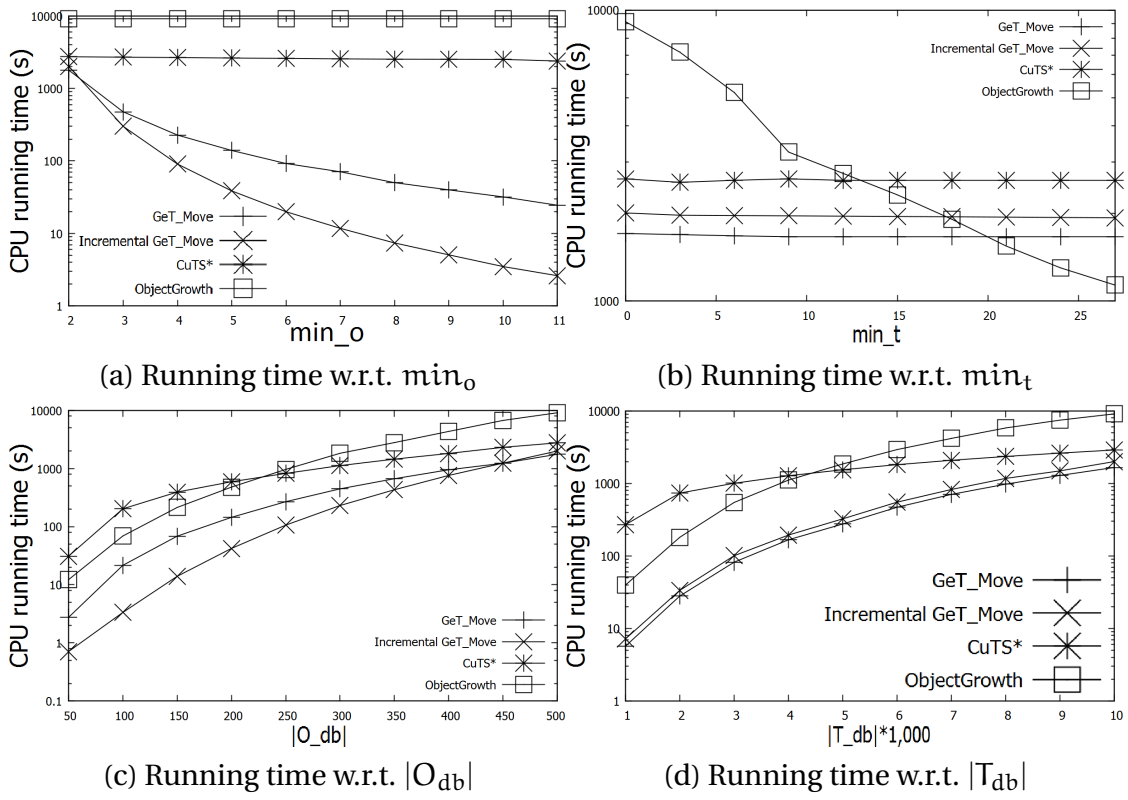
(a) Running time w.r.t. $\mathtt{min}_o$ (b) Running time w.r.t. $\mathtt{min}_t$

(c) Running time w.r.t. $|O_{db}|$ (d) Running time w.r.t. $|T_{db}|$

Figure 3.7: Running time on Buffalo dataset.

includes 50,000 objects moving during 10,000 timestamps and it contains 500 million locations in total. The executions of CMC and CuTS* stop due to a lack of memory capacity after processing 300 milion locations. Additionally, ObjectGrowth cannot provide the results after 1day running. The main reason is that with low $\mathtt{min}_t$ ($=1$), the search space is significant larger ($\approx 2^{50,000}$). Meanwhile, thanks to the LCM approach, our algorithms can provide the results within hours (Figure 3.12).

Efficiency w.r.t. *Block-size.* To investigate the optimal value of block-size, we examine Incremental GeT_Move by using the default values of $\mathtt{min}_o, \mathtt{min}_t$ with different block-size values on real datasets and synthetic dataset ($|O_{db}| = 500, |T_{db}| = 1{,}000$). The optimal block-size range can be from 20 to 30 timestamps within which Incremental GeT_Move obtains the best performance for all the datasets (Figure 3.13). This is because objects tend to move together in suitable short interval (from 20 to 30 timestamps). Therefore, by setting the block-size in this range, the data is efficiently compressed into FCIs. Meanwhile, with larger block-size values, the objects' movements are quite different; therefore, the data compressing is not so efficient. Regarding to small block-size values (from 5 to 15), we have to face up to a large number of blocks so that the process is slowed down. In the previous experiments, block-size is set to 25.



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

(c) Running time w.r.t. $|O_{db}|$ (d) Running time w.r.t. $|T_{db}|$

Figure 3.8: Running time on Synthetic dataset.

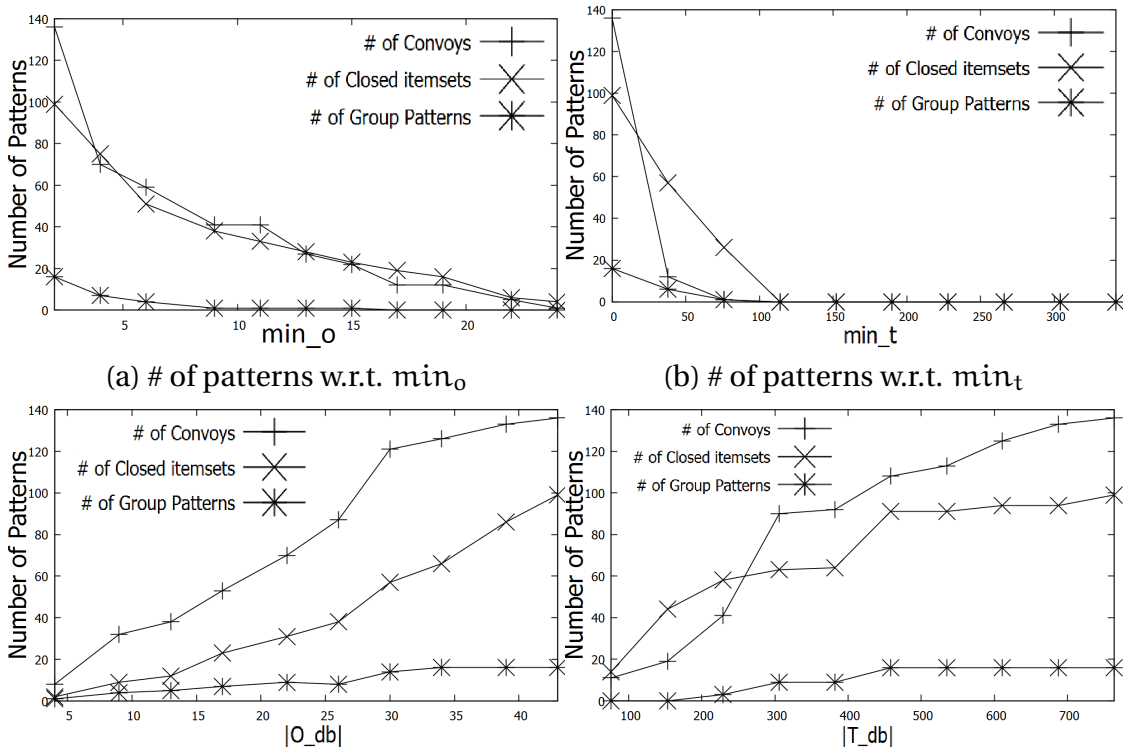
3.3.3 Toward A Parameter Free Incremental GeT_Move Algorithm

Until now, we have presented the Incremental GeT_Move which split the original cluster matrix into different equivalent blocks. The experiment results show that the algorithm is efficient. However, the disadvantage of this approach is that we do not know what is the optimal block size. To identify the optimal block sizes, different techniques can be applied, such as data sampling in which a sample of data is used to investigate the optimal block sizes. Even if this approach is appealing, extracting such a sample is very difficult.

To tackle this problem, we propose an innovative solution to dynamically assign blocks to Incremental GeT_Move. First of all, we would like to propose the definition of a fully nested cluster matrix (resp. block) (Figure 3.14c) as follows.

Definition 9. *Fully nested cluster matrix (resp. block). An* $n \times m$ *0-1 block* b *is fully nested if for any two column* r_i *and* r_{i+1} $(r_i, r_{i+1} \in b)$ *we have* $r_i \cap r_{i+1} = r_{i+1}$.

We can consider that the *LCM* is very efficient when it is applied on dense (resp. (fully) nested) datasets and blocks. Let E be the universe of items, consisting of items $1, \ldots, n$. A subset X of E is called an itemset. In the LCM algorithm process on a common cluster



(a) # of patterns w.r.t. min_o (b) # of patterns w.r.t. min_t

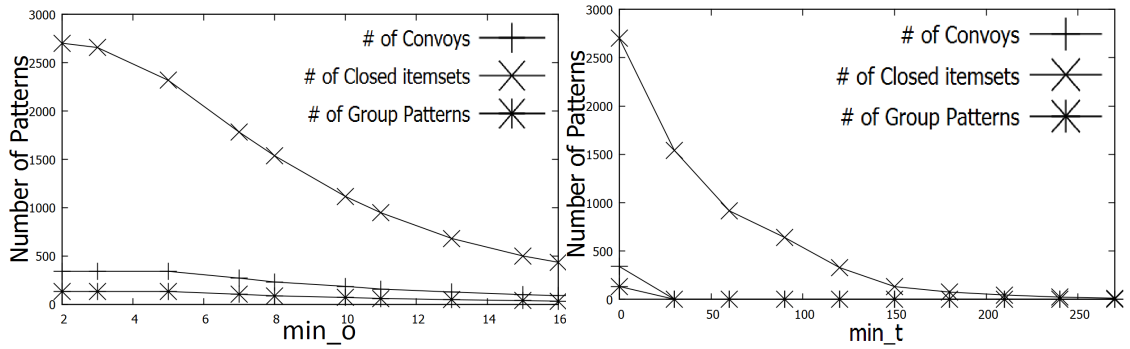
(c) # of patterns w.r.t. $|O_{db}|$ (d) # of patterns w.r.t. $|T_{db}|$

Figure 3.9: Number of patterns on Swainsoni dataset. Note that # of frequent closed itemsets is equal to # of closed swarms.

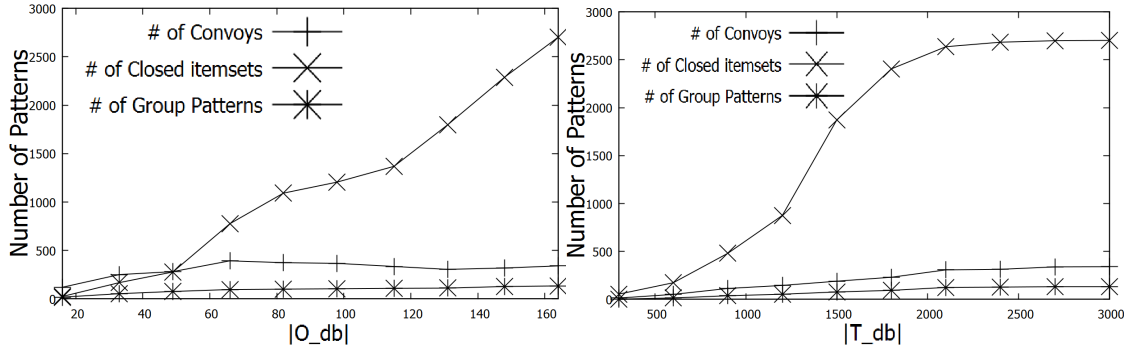
matrix, for any X, we make the recursive call for $X[i]$ for each $i \in \{i(X)+1,\ldots,|E|\}$ because we do not know which $X[i]$ will be a closed itemset when X is extended by adding i to X. Meanwhile, for a fully nested cluster matrix, we know that only the recursive call for item $i = i(X)+1$ is valuable and the other recursive calls for each item $i \in \{i(X)+2,\ldots,|E|\}$ are useless. Note that $i(X)$ returns the last item of X.

Property 6. *Recursive Call. Given a fully nested cluster matrix* nCM *(resp. block), a universe of items* E *of* nCM*, an itemset* X *which is a subset of* E*. All the FCIs can be generated by making a recursive call of item* $i = i(X)+1$.

Proof. After construction, we have $\forall i \in E, O(i) \cap O(i+1) = O(i+1)$; thus, $O(i+1) \subseteq O(i)$. Additionally, $\forall i' \in \{i(X)+2,\ldots,|E|\}$ we need to make a recursive call for $X[i']$ and let assume that we obtain a frequent itemset $X \cup i' \cup X'$ with $X' \subseteq \{i(X)+3,\ldots,|E|\}$. We can consider that $O(i') \subseteq O(i(X)+1)$ and therefore $O(X \cup i' \cup X') = O(X \cup (i(X)+1) \cup i' \cup X')$. Consequently, $X \cup i' \cup X'$ is not a FCI because $(X \cup i' \cup X') \subset (X \cup (i(X)+1) \cup i' \cup X')$ and $O(X \cup i' \cup X') =$



(a) # of patterns w.r.t. min_o (b) # of patterns w.r.t. min_t



(c) # of patterns w.r.t. $|O_{db}|$ (d) # of patterns w.r.t. $|T_{db}|$

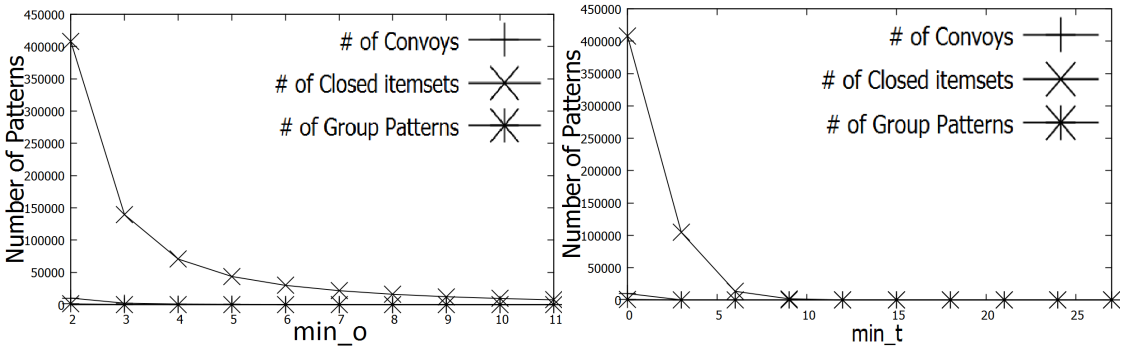
Figure 3.10: Number of patterns on Buffalo dataset. Note that # of frequent closed itemsets is equal to # of closed swarms.

$O(X \cup (i(X)+1) \cup i' \cup X')$. Furthermore, $(X \cup (i(X)+1) \cup i' \cup X')$ can be generated by making a recursive call for $i(X)+1$. We can conclude that it is useless to make a recursive call for $\forall i' \in \{i(X)+2,\ldots,|E|\}$ and additionally, all FCIs can be generated only by making a recursive call for $i(X)+1$. □

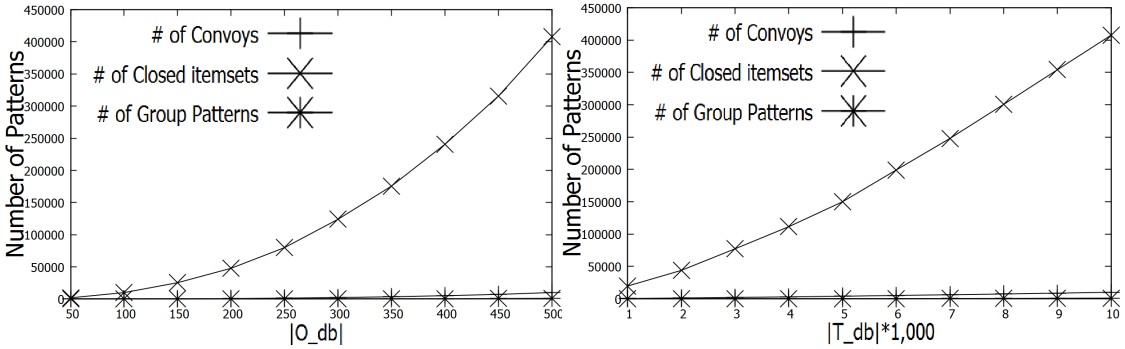
By applying *Property 6*, we can consider that LCM is more efficient on a fully nested matrix because it reduces unnecessary recursive calls. Therefore, our goal is to retrieve fully nested blocks to improve the performance of Incremental GeT_Move. In order to reach this goal, we first apply the *nested and segment nested Greedy* algorithm [6] ([37]) to re-arrange the cluster matrix (Figure 3.14a) so that it now becomes a *nested cluster matrix* (Figure 3.14b). Then, we propose a sub-function *Nested Block Partition* (Figure 3.3-(4)) to dynamically split the nested cluster matrix into *fully nested blocks* (Figure 3.14c).

By following the Definition 9 and scanning the nested cluster matrix from the beginning to the end, we are able to obtain all fully nested blocks. We start from the first column of nested cluster matrix, then we check the next column and if the nested condition is held

6. http://www.aics-research.com/nestedness/



(a) # of patterns w.r.t. min_o (b) # of patterns w.r.t. min_t



(c) # of patterns w.r.t. $|O_{db}|$ (d) # of patterns w.r.t. $|T_{db}|$

Figure 3.11: Number of patterns on Synthetic dataset. Note that # of frequent closed itemsets is equal to # of closed swarms.

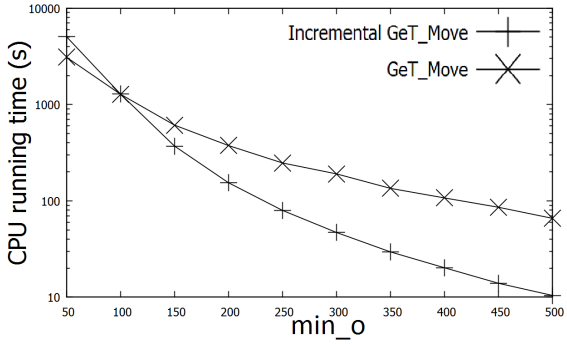


Figure 3.12: Running time w.r.t min_o on large Synthetic dataset.

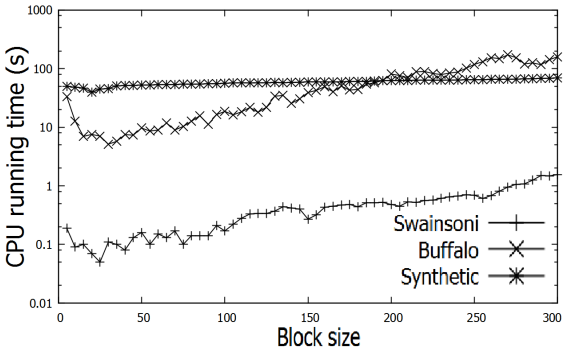


Figure 3.13: Running time w.r.t block size.

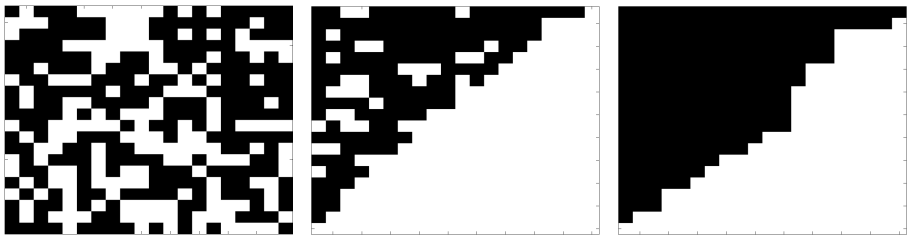
Algorithm 3: Fully Nested Block Partition

Input : a nested cluster matrix CM_N
Output: a set of blocks B

```
1  begin
2      B := ∅; NestedB := ∅; SpareB := ∅;
3      foreach item i ∈ CM_N do
4          if i ∩ (i+1) = (i+1) then
5              NestedB := NestedB ∪ i;
6          else
7              NestedB := NestedB ∪ i;
8              if |NestedB| ≤ 1 then
9                  SpareB.push_all(NestedB);
10                 NestedB := ∅
11             else
12                 B := B ∪ NestedB;
13                 NestedB := ∅
14     return B := B ∪ SpareB;
15 where the purpose SpareB.push_all(NestedB) function is to put all items in
   NestedB to SpareB.
```

then the block is expanded; otherwise, the block is set and we create a new block. Note that all small blocks containing only 1 column are merged into a sparse block $SpareB$. At the end, we obtain a set of fully nested blocks $NestedB$ and a sparse block $SpareB$. Finally, the Incremental GeT_Move is applied on $B = NestedB \cup SpareB$.

The pseudo code of Fully Nested Block Partition sub-function is described in *Algorithm 3*.



(a) (b) (c)

Figure 3.14: Examples of non-nested , almost nested, fully nested datasets [37]. Black = 1, white = 0. (a) Original, (b) Almost nested, (c) Fully nested.

Table 3.4: An example of FCI binary presentation.

		$FCIs_{db}$			$FCIs_{db'}$	
$binary(FCI)$		$b(ci_1)$	$b(ci_2)$		$b(ci'_1)$	$b(ci'_2)$
O_{db}	o_1	1	0		1	0
	o_2	1	0		1	1
	o_3	0	1		0	1
	o_4	0	1		0	1

3.3.4 Object Movement Pattern Mining Algorithm Based on Explicit Combination of FCI Pairs

In real world applications (e.g. cars), object locations are continuously reported by using Global Positioning System (GPS). Therefore, new data is always available. Let us denote the new movement data as $(O_{db}, T_{db'})$. Naturally, it is cost-prohibitive and time consuming to execute Incremental GeT_Move (or GeT_Move) on the entire database (denoted $(O_{db}, T_{db} \cup T_{db'})$) which is created by merging $(O_{db}, T_{db'})$ into the existing database (O_{db}, T_{db}). To tackle this issue, we provide an approach which efficiently combines the existing frequent closed itemsets $FCIs_{db}$ with the new frequent closed itemsets $FCIs_{db'}$, which are extracted from db', to obtain the final results $FCIs_{db \cup db'}$.

For instance, in Table 3.4, we have two sets of frequent closed itemsets $FCIs_{db}$ and $FCIs_{db'}$. Each FCI will be presented as a $|O_{db}|$-bit binary numeral. Let us define a set of operations that will be used for object movement pattern mining based on explicit combination of FCI pairs. Given two FCIs ci and ci', we have that:

- $ci \wedge ci'$: returns $b(ci) \wedge b(ci')$.
- $ci \vee ci'$: returns $b(ci) \vee b(ci')$.
- $ci \cup ci'$: returns a set of clusters that are the union of ci_1 and ci'_1.
- $Size(ci)$ returns the number of '1's in ci. Note that $Size(ci) = O(ci) = \sigma(ci)$.

The principle function of our algorithm is to explicitly combine all pairs of $FCIs(ci, ci')$ to generate new FCIs. Let us assume that $ci \wedge ci' = \gamma$, $\gamma = ci \cup ci'$ is a FCI if $\sigma(\gamma)$ is larger

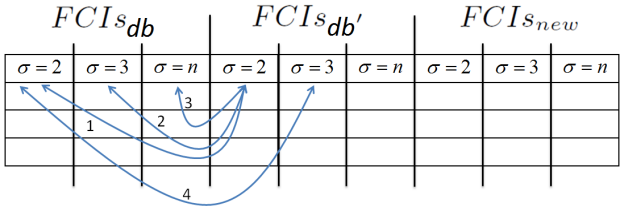


Figure 3.15: An example of the explicit combination of pairs of FCIs-based approach.

than min_o and that there are no subsets of $O(ci)$, $O(ci')$ so that they are supersets of $O(\gamma)$. Here is an explicit combination of a pair of $FCIs(ci, ci')$:

Property 7. *Explicit Combination of a pair of FCIs. Given FCIs* ci *and* ci' *so that* $ci \in FCIs_{db}, ci' \in FCIs_{db'}$, *a* . $ci \cup ci'$ *is a FCI that belongs to* $FCIs_{db \cup db'}$ *if and only if:*

$$\begin{cases} \textit{if } ci \wedge ci' = \gamma \textit{ then} \\ (1): Size(\gamma) \geq min_o. \\ (2): \nexists p: p \in FCIs_{db}, O(\gamma) \subseteq O(p) \subseteq O(ci). \\ (3): \nexists p': p' \in FCIs_{db'}, O(\gamma) \subseteq O(p') \subseteq O(ci'). \end{cases} \tag{3.5}$$

where $ci = \{c_1, c_2, \ldots, c_p\}$ *and* $ci' = \{c'_1, c'_2, \ldots, c'_p\}$.

Proof. After construction, we have $\nexists p: p \in FCIs_{db}, O(\gamma) \subseteq O(p) \subseteq O(ci)$. We assume that $\exists i$ s.t. $i \in C_{db}, O(\gamma) \subseteq i$ and $i \notin ci$ therefore $\exists p$ s.t. $p = \{\forall i | i \in C_{db}, O(\gamma) \subseteq i, i \notin ci\} \cup ci, O(\gamma) \subseteq O(p)$. Consequently, $\forall i \in C_{db}, O(\gamma) \subseteq i$ then $i \in p$ and therefore p is a FCI and $p \in FCIs_{db}$. This violates the assumption and therefore $\nexists i$ s.t. $i \in C_{db}, O(\gamma) \subseteq i$ and $i \notin ci$ or $\forall i$ s.t. $i \in C_{db}, O(\gamma) \subseteq i$ then $i \in ci$. Similarly, if $\nexists p': p' \in FCIs_{db'}, O(\gamma) \subseteq O(p') \subseteq O(ci')$ then $\forall i'$ s.t. $i' \in C_{db'}, O(\gamma) \subseteq i'$ then $i' \in ci'$. Consequently, if $\forall i \in C_{db \cup db'}, O(\gamma) \subseteq i$ then $i \in ci \cup ci'$. Additionally, $Size(\gamma) = \sigma(\gamma) \geq min_o$ and therefore $ci \cup ci'$ is a FCI and $ci \cup ci' \in FCIs_{db \cup db'}$. □

We can consider that if $ci \cup ci'$ is a FCI, they must respectively be the two longest FCIs which contain $O(\gamma)$ in $FCIs_{db}$ and $FCIs_{db'}$. $(O(\gamma), ci \cup ci')$ is a new FCI and it will be stored in a set of new frequent closed itemsets, named $FCIs_{new}$. To efficiently make all combinations, we first partition $FCIs_{db}, FCIs_{db'}$ and $FCIs_{new}$ into different partitions in terms of support so that the FCIs, that have the same support value, will be in the same partition (Figure 3.15). Secondly, partitions are combined from the smallest support values (resp. longest FCIs) to the largest ones (resp. shortest FCIs). New FCIs will be added into the right partition in $FCIs_{new}$. By using this approach, it is guaranteed that the first time there is $ci \wedge ci' = \gamma, Size(\gamma) \geq min_o$ then $ci \cup ci'$ is a new FCI because they are the two longest FCIs which contain $O(\gamma)$. Therefore, we just ignore the later combinations which return γ as the result. Furthermore, to ensure that γ already exists in $FCIs_{new}$ or not, we only need to check items in the $FCIs_{new}$ partition whose support value is equal to

$Size(\gamma)$. We can consider that by partitioning $FCIs_{db}, FCIs_{db'}$ and $FCIs_{new}$, the process is much improved. Additionally, we also propose a pruning rule to speed up the approach by ending the combination running of a FCI ci' as follows:

Lemma 4. *The combination running of* ci' *is ended if:*

$$\exists ci \in FCIs_{db} \text{ s.t. } ci \wedge ci' = ci', ci \cup ci' \text{ is a FCI.} \tag{3.6}$$

Proof. Assume that $\exists \Upsilon : \Upsilon \in FCIs_{db}, \sigma(\Upsilon) \geq \sigma(ci), \Upsilon \wedge ci' = ci'$. If $O(ci) \subseteq O(\Upsilon)$ then we have $ci \in FCIs_{db}, O(ci') \subseteq O(ci) \subseteq O(\Upsilon)$ and this violates the condition 2 in *Property 7*, therefore $\Upsilon \cup ci'$ is not a FCI. If $O(ci) \nsubseteq O(\Upsilon)$ then $\exists i \in C_{db}$ s.t. $O(ci') \subseteq i$ and $i \notin \Upsilon$. Furthermore, $\exists p : p = \{\forall i | i \in C_{db}, O(ci') \subseteq i, i \notin \Upsilon\} \cup \Upsilon$. So, $\forall i, i \in C_{db}, O(ci') \subseteq i$ then $i \in p$ and therefore p is a FCI and $p \in FCIs_{db}$. Additionally, $O(ci') \subseteq O(p) \subseteq O(\Upsilon)$. This violates the condition 2 in *Property 7*, therefore $\Upsilon \cup ci'$ is not a FCI. Consequently, we can conclude that $\nexists \Upsilon$ s.t. $\Upsilon \in FCIs_{db}, \sigma(\Upsilon) \geq \sigma(ci), \Upsilon \wedge ci' = ci'$ and $\Upsilon \cup ci'$ is a FCI. Therefore, we do not need to continue the combination running of ci'. □

Similar to Lemma 4, in the explicit combination process, ci will be deactivated for further combinations when there is a ci' so that $ci \wedge ci' = ci$ and $ci \cup ci'$ is a FCI. After generating all new FCIs in $FCIs_{new}$, the final results $FCIs_{db \cup db'}$ is created by collecting FCIs in $FCIs_{db}, FCIs_{db'}, FCIs_{new}$. In this step, some of them will be discarded such that:

Property 8. *Discarded FCIs in* $FCIs_{db \cup db'}$ *creating step. All the FCIs which satisfy the following conditions will not be selected as a FCIs in the final results.*

$$\begin{cases} (1) : \forall ci \in FCIs_{db}, \textit{ if } \exists ci' \in FCIs_{db'} \textit{ s.t. } ci \wedge ci' = ci. \\ (2) : \forall ci' \in FCIs_{db'}, \textit{ if } \exists ci \in FCIs_{db} \textit{ s.t. } ci \wedge ci' = ci'. \end{cases} \tag{3.7}$$

Note that during the explicit combination step, the FCIs which will not be selected for the final results are removed by applying Property 8. It means that we only add all suitable FCIs into $FCIs_{db \cup db'}$ and therefore it is optimized and much less costly. In the worst case scenario, the complexity of explicit combination of pairs of FCIs step is $O(|FCIs_{db}| \times |FCIs_{db'}| \times \frac{|FCIs_{new}|}{\#partitions(FCIs_{new})})$. Naturally, $T_{db'}$ is much smaller than T_{db} and therefore $FCIs_{db'}, FCIs_{new}$ are very small compare to $FCIs_{db}$. Consequently, the process can be potentially greatly improved when compare to the executing of Incremental GeT_Move on the entire database $(O_{db}, T_{db \cup db'})$.

The pseudo code of the *Object Movement Pattern Mining Algorithm Based on Explicit Combination of FCI Pairs* is described in *Algorithm 4*.

Algorithm 4: Explicit Combination of Pairs of FCIs-based Object Movement Pattern Mining Algorithm

Input : a set of FCIs $FCIs_{db}$, Occurrence sets K, int min_o, int min_t, double θ, set of Occurrence sets (blocks) B', int min_c, double min_{wei}

1 **begin**
2 $FCIs_{db'} := \emptyset; FCIs_{new} := \emptyset; FCIs_{db \cup db'} := \emptyset;$
3 $FCIs_{db'}$:= Incremental GeT_Move* ($K, min_o, min_t, CI, \theta, B', min_c, min_{wei}$);
4 **foreach** *partition* $P' \in FCIs_{db'}$ **do**
5 **foreach** *FCI* $ci' \in P'$ **do**
6 **foreach** *partition* $P \in FCIs_{db}$ **do**
7 **foreach** *FCI* $ci \in P$ **do**
8 $\gamma := ci \wedge ci';$
9 **if** $Size(\gamma) \geq min_o$ *and* $FCIs_{new}.notContain(\gamma, Size(\gamma))$ **then**
10 $\gamma := ci \cup ci';$
11 $FCIs_{new}.add(\gamma, Size(\gamma));$
12 **if** $\gamma = ci$ **then**
13 $FCIs_{db}.remove(ci);$
14 **if** $\gamma = ci'$ **then**
15 $FCIs_{db}.remove(ci');$
16 **go to** line 5;
17 $FCIs_{db \cup db'} := FCIs_{db} \cup FCIs_{db'} \cup FCIs_{new};$
18 **foreach** *FCI* $X \in FCIs_{db \cup db'}$ **do**
19 **PatternMining**(X, min_t); /*X is a pattern?*/
20 Where: Incremental GeT_Move* is an Incremental GeT_Move without *PatternMining* sub-function, $FCIs_{new}.notContain(\gamma, Size(\gamma))$ returns true if there does not exists γ in partition which has the support value is $Size(\gamma)$.

3.4 Experimental Results

3.4.1 Parameter Free Incremental GeT_Move Efficiency

The experimental results show that, so far, Incremental GeT_Move and GeT_Move outperform other algorithms. Additionally, our algorithms can work with low min_o and min_t values. In this section, we perform another experiment to examine the efficiency of the Parameter Free Incremental GeT_Move algorithm. In this experiment, we compare performances of four algorithms: 1) Parameter free Incremental GeT_Move, named Nested Incremental GeT_Move, 2) Nested GeT_Move which is the application of GeT_Move on nested cluster matrix CM_N, 3) Incremental GeT_Move which is executed with the optimal block size values on original cluster matrix CM, 4) GeT_Move which is applied on original cluster matrix CM.

Table 3.5: Fully nested blocks on datasets.

Dataset	Matrix fill	#Nested blocks	avg.length
Swainsoni	17.8%	102	4.52
Buffalo	7.2%	602	2.894
Synthetic	0.1%	8	2.00

Efficient w.r.t. *Real datasets.* Figures 3.17, 3.18 show that Nested Incremental GeT_Move (resp. Parameter Free Incremental GeT_Move) greatly outperforms the other algorithms. It is due to the better performance of LCM algorithm on nested cluster matrix (resp. fully nested blocks) compared to the original cluster matrix. Essentially, with the nested cluster matrix, the number of combinations of frequent itemsets X and items i to ensure the closeness is greatly reduced. Therefore, the performance of the LCM algorithm is greatly improved. The fact is Nested GeT_Move is always better than GeT_Move (Figures 3.17, 3.18, 3.19). Additionally, the Swainsoni and Buffalo datasets contain many fully nested blocks (Table 3.5 and Figure 3.16). Consequently, the Nested Incremental GeT_Move is more efficient than the other algorithms.

Efficient w.r.t. *Synthetic dataset.* We can consider that Nested Incremental GeT_Move is quite similar to Nested GeT_Move (Figure 3.19). This is because: 1) Synthetic data is very sparse, 2) there are few fully nested blocks, 3) the nested blocks contain a very small number of items (i.e. 0.1% matrix fill by '1' and only 8 fully nested blocks which average length is 2, see Table 3.5 and Figures 3.16e-f). Therefore, the processing time of nested blocks is quite short. Meanwhile, there is a large nested sparse block which is the main partition that need to be processed by both Nested Incremental GeT_Move and Nested GeT_Move.

Additionally, thanks to the nested sparse block, the performance of LCM is improved a lot. Therefore, Nested Incremental GeT_Move and Nested GeT_Move are better than the others in most of cases. Exceptionally, with small number of objects $|O_{db}|$ (i.e. $|O_{db}| = 50$, Figure 3.19c) or high min_o (i.e. $min_o \geq 9$, Figure 3.19a), Incremental GeT_Move is slightly better than Nested Incremental GeT_Move and Nested GeT_Move. The main reason is that Incremental GeT_Move splits the cluster matrix CM into different small blocks within which there are a small number of items and FCIs. Thus, the computation cost is reduced. On the other hand, Nested Incremental GeT_Move and Nested GeT_Move need to work with a large nested sparse block.

3.4.2 Movement Pattern Mining Algorithm Based on Explicit Combination of FCI Pairs

In this section, an experiment is designed to examine the object movement pattern mining algorithm based on explicit combination of FCI pairs and to identify when we

(a) Original Swainsoni cluster matrix.

(b) Nested Swainsoni cluster matrix.

(c) Original Buffalo cluster matrix.

(d) Nested Buffalo cluster matrix.

(e) Original Synthetic cluster matrix.

(f) Nested Synthetic cluster matrix.

Figure 3.16: Original cluster matrices and nested cluster matrices.

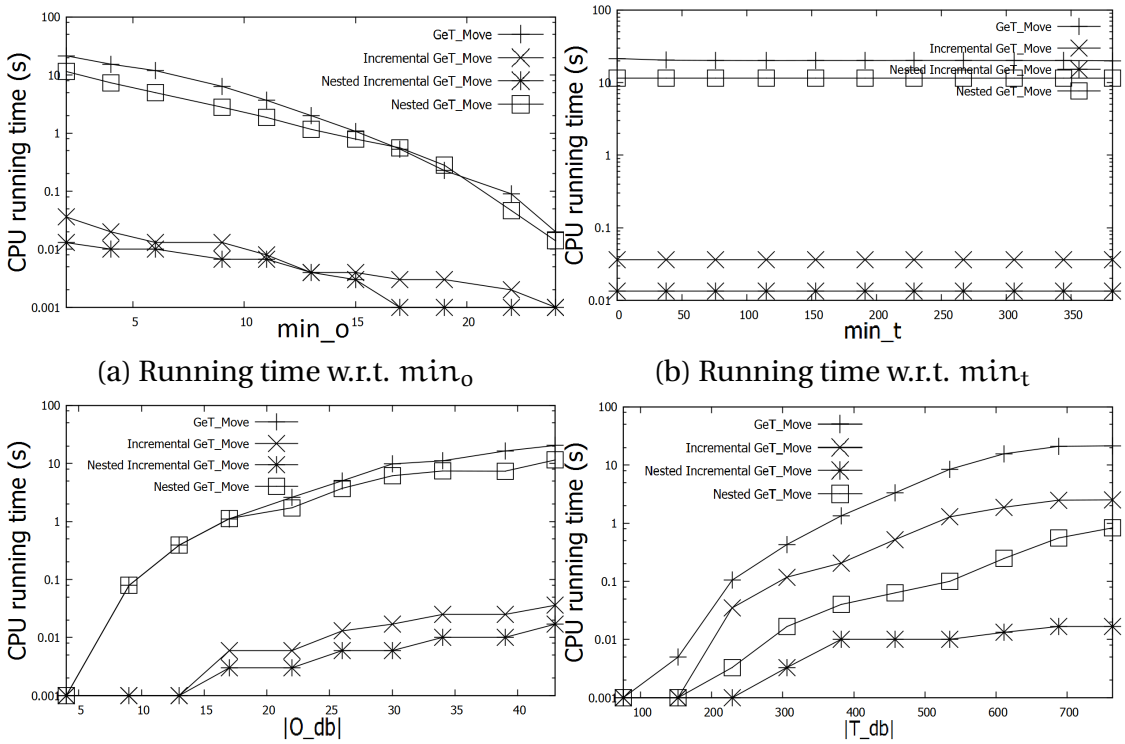
should update the database. We first use half of Swainsoni, Buffalo and Synthetic datasets as a db. Then the other half is used to generate db′ which is increased step by step up to the maximum size (Figure 3.20). In this experiment, Incremental GeT_Move is employed to extract FCIs from db and db′.

For real datasets (Swainsoni and Buffalo), the explicit combination algorithm is more efficient than the Incremental GeT_Move in all cases (Figures 3.20a, b). This is because we already have $FCIs_{db}$ and therefore we only need to extract $FCIs_{db'}$ and then combine $FCIs_{db}$ and $FCIs_{db'}$. Additionally, the Swainsoni and Buffalo are sufficiently dense (i.e. 17.8% and 7.2% with large number of fully nested blocks, see Table 3.5) so that the numbers of FCIs in $FCIs_{db}$ and $FCIs_{db'}$ are not huge. Consequently, the number of combinations is reduced and thus the algorithm is more efficient. In Figures 3.20a-b, we can consider that the running time of the explicit combination algorithm significantly changes when $|T_{db'}| > 15\%|T_{db}|$. This means that it is better to update the database when $|T_{db'}| < 15\%|T_{db}|$.

For the synthetic dataset, the explicit combination algorithm is only efficient on small db′ (i.e. $|T_{db'}| < 20\%|T_{db}|$, Figure 3.20c) because the dataset is very sparse. In fact, the number of FCIs in $FCIs_{db'}$ is enlarged when the size of db′ increases. Thus, the explicit combination algorithm is not efficient because of the huge number of combinations.

Overall, we can consider that the explicit combination algorithm obtains good efficiency when $T_{db'}$ is smaller than 15% of T_{db}.

To summarize, Incremental GeT_Move and GeT_Move outperform the other algo-



(a) Running time w.r.t. $\mathtt{min}_o$

(b) Running time w.r.t. $\mathtt{min}_t$

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

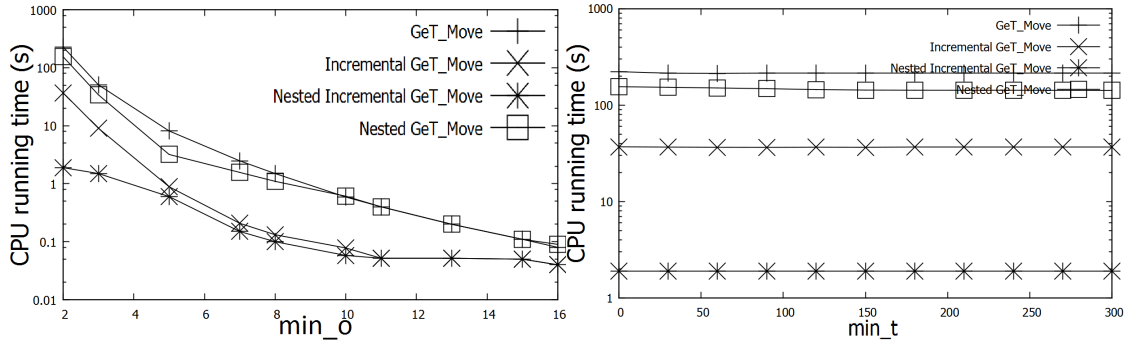
(d) Running time w.r.t. $|T_{db}|$

Figure 3.17: Running time on Swainsoni dataset.

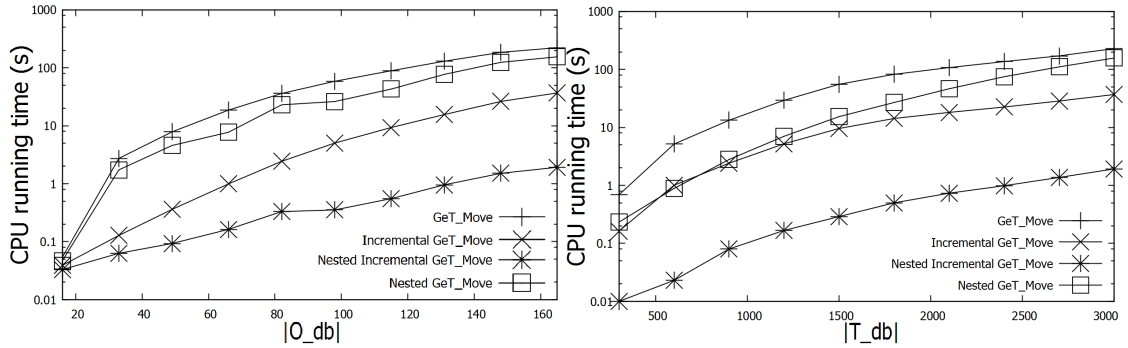
rithms. Additionally, our algorithms can work with low values of $\mathtt{min}_o$ and $\mathtt{min}_t$. To reach the optimal efficiency, we propose a parameter free Incremental GeT_Move (resp. Nested Incremental GeT_Move) which dynamically assigns fully nested blocks for the algorithm from the nested cluster matrix. The experimental results show that the efficiency is greatly improved with the Nested Incremental GeT_Move and Nested GeT_Move. Furthermore, by storing FCIs in a closed itemset database (see Figure 3.3), it is possible to reuse them whenever new object movements arrive. The experimental results show that it is better to update the database when $T_{db'}$ is smaller than 15% of T_{db} by applying the explicit combination algorithm.

3.5 Discussion

In this chapter, we propose a (parameter free) unifying incremental approach to automatically extract different kinds of object movement patterns by applying frequent closed itemset mining techniques. Their effectiveness and efficiency have been evaluated by us-



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



(c) Running time w.r.t. $|O_{db}|$ (d) Running time w.r.t. $|T_{db}|$

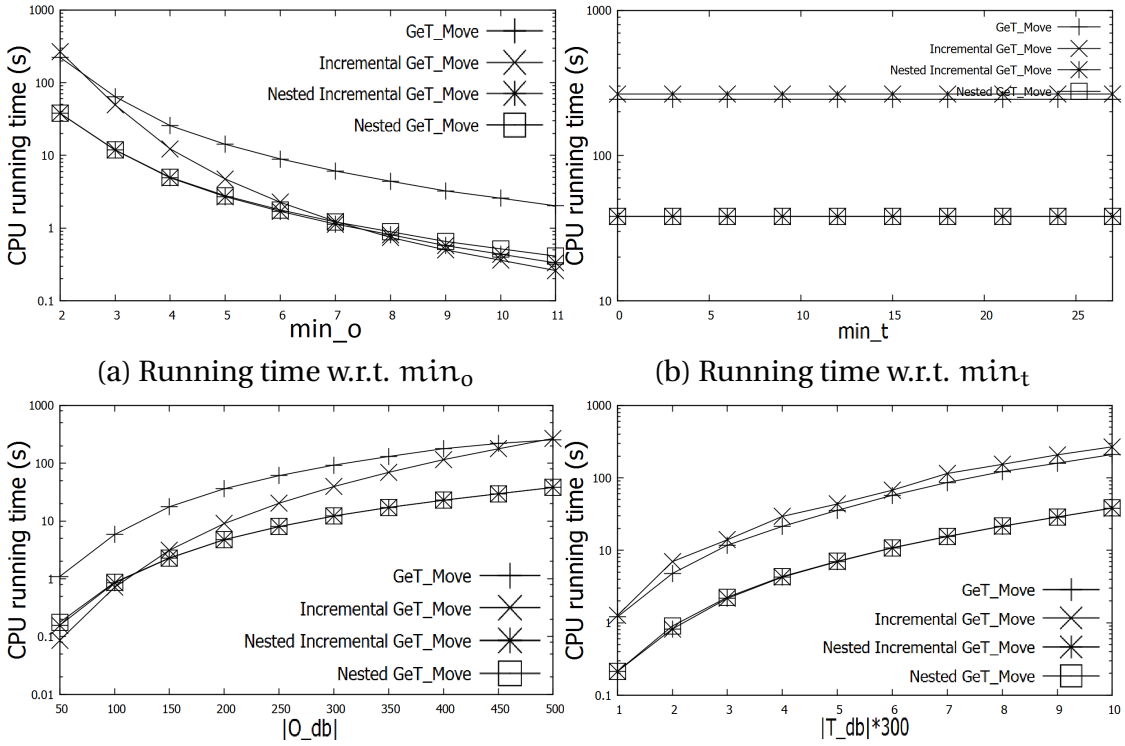
Figure 3.18: Running time on Buffalo dataset.

ing real and synthetic datasets. Experiments show that our approaches outperform traditional ones.

Now, as we have seen, we can store the results (resp. FCIs) to improve the process when new object movements arrive. 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.

Even GeT_Move can help us extract and manage different kinds of patterns. The challenges are still remained. Indeed, in some real world cases, the existing pattern models are not relevant enough to help us fully understand the behavior of moving objects. For instance, they require the group of moving objects to be together for at least min_t timestamps, i.e. could be consecutive or completely be non-consecutive, which might not be practical in the real cases. Enforcing consecutive time constraint may results in loosing meaningful patterns while completely relax this constrain may generate large amount of extraneous patterns.

To deal with this issue, we propose the *fuzzy moving object cluster* concept which will be clearly presented in the next Chapter.



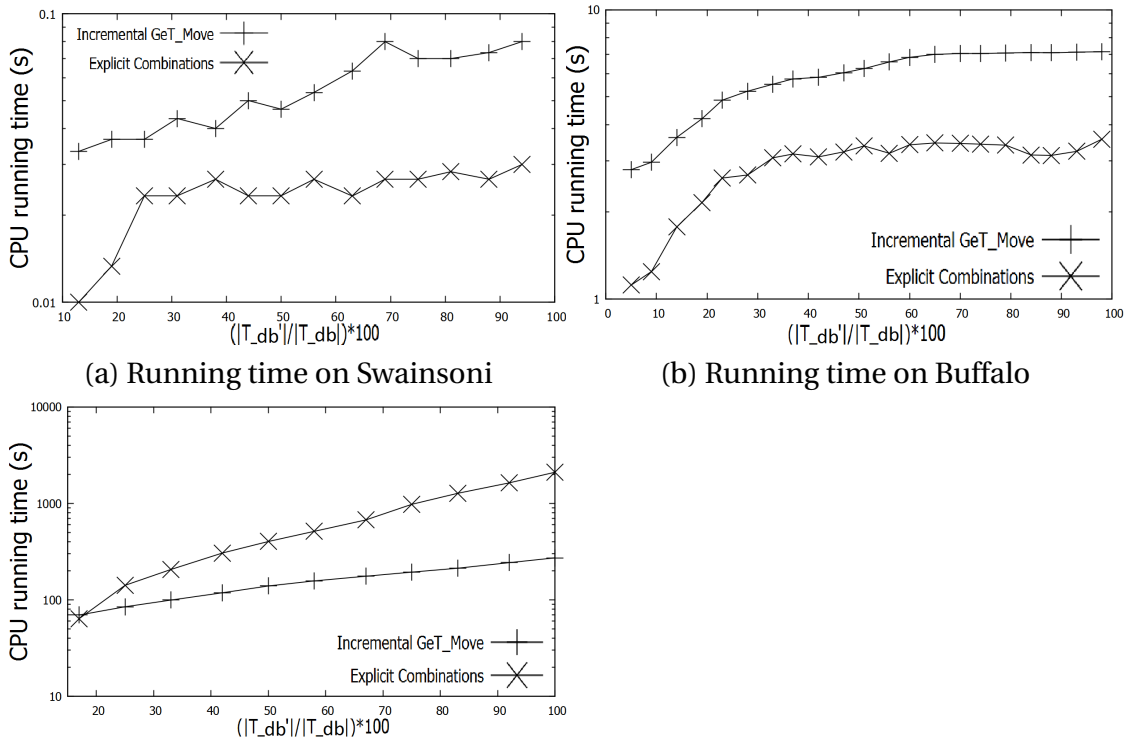
(a) Running time w.r.t. min_o

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

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

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

Figure 3.19: Running time on Synthetic dataset.



(a) Running time on Swainsoni

(b) Running time on Buffalo

(c) Running time on Synthetic data

Figure 3.20: Explicit combination algorithm efficiency.

CHAPTER

4 Mining Fuzzy Moving Object Clusters

Preamble

Naturally, the moving objects in a cluster may actually diverge temporarily and congregate at certain timestamps. Thus, there are time gaps among moving object clusters. Existing approaches either put a strong constraint (i.e. no time gap) or completely relaxed (i.e. whatever the time gaps) in dealing with the gaps may result in the loss of interesting patterns or the extraction of huge amounts of extraneous patterns. Thus it is difficult for analysts to understand the object movement behavior.

Motivated by this issue, we propose the concept of fuzzy swarm which softens the time gap constraint. The goal of this chapter is to find all non-redundant fuzzy swarms, namely fuzzy closed swarm. As a contribution, we propose fCS-Miner algorithm which enables us to efficiently extract all the fuzzy closed swarms. Conducted experiments on real and large synthetic datasets demonstrate the effectiveness, parameter sensitiveness and efficiency of our methods.

4.1 Introduction

Let us consider the Figure 4.1, if we set $\mathtt{min}_t = 5$ and the timestamps must be consecutive, no moving object cluster can be found. But essentially, these two objects, o_3 and o_4, travel together even though they temporarily leave the cluster at some snapshots. To address this issue, Zhenhui Li et al. [23] propose swarm in which moving objects are not required to be together in consecutive timestamps. Therefore, swarm can capture the movement pattern. The pattern is *"o_3, o_4 are moving together from* A *to* C *to* D *to* E *and to* H *at timestamps* t_1, t_2, t_3, t_4 *and* t_{999}*"*. This pattern could be interesting since it expresses

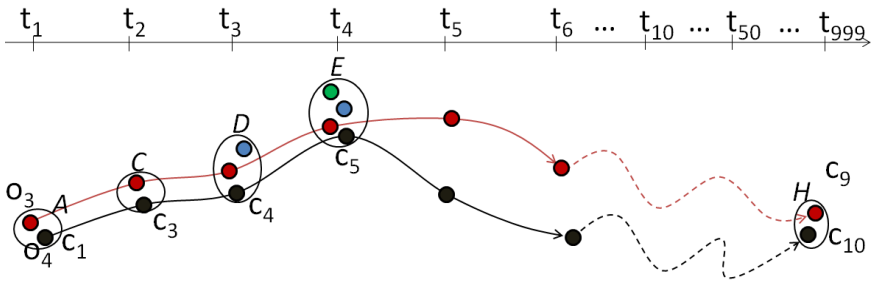


Figure 4.1: An example of moving object clusters. o_3, o_4 are moving objects, $c_1, \ldots, c_5, c_{10}$ are clusters which are generated by applying some clustering techniques and *A, C, D, E, H* are spatial regions.

the relationship between o_3 and o_4. However, the issue here is that it is hard to say that o_3 and o_4 moving together to F since they only meet each other at F by chance after 995 timestamps. In other words, enforcing the consecutive time constraint may result in the loss of interesting moving object clusters, while completely relaxing this constraint may generate a large number of extraneous and useless patterns.

In this chapter, we propose a new movement pattern, called *fuzzy closed swarm*, which softens the consecutive time constraint without generating extraneous patterns. The key challenge is to deal with the time gap between a pair of clusters since: 1) it is difficult to recognize which size of a time gap is relevant or not, 2) we need to know when the patterns should be ended to eliminate uninteresting ones. To address these issues, we present the definition of *fuzzy time gap* and *fuzzy time gap participation index*. Obtained patterns are of the type *"o_1, o_2 are moving together from A to B to C with 60% weak, 20% medium and 20% strong time gaps"*. These patterns are characterized by their time gap frequency (or support), which is by definition the proportion of time gaps involved in the patterns. As a contribution, we propose *fCS-Miner* algorithm to efficiently extract the complete set of fuzzy closed swarms. The approach shares the same spirit with the GeT_Move algorithm [12] [13] but is different in terms of goal and properties. The effectiveness as well as efficiency of our method are demonstrated on both real and large scale synthetic moving object databases.

This chapter is structured as follows. Section 2 discusses the related work. The definitions of fuzzy time gap and fuzzy closed swarm are given in Section 3. fCS-Miner algorithm will be clearly presented in Section 4. Experiments testing effectiveness and efficiency are shown in Section 5. Finally, we draw our conclusions in Section 6.

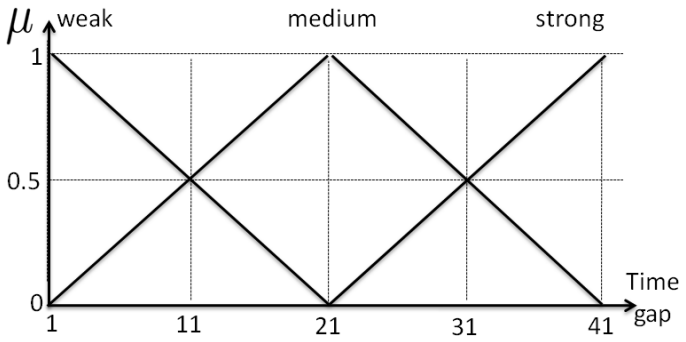


Figure 4.2: Membership degree functions for fuzzy time gaps.

4.2 Fuzzy Closed Swarms

In this section, we will clearly present the definition of fuzzy closed swarm. In order to do so, we first propose the adaptation of fuzzy logic principle in which the strength of time gaps are evaluated with a membership degree function A, see Figure 4.2. Given two timestamps t_1 and t_2, a time gap x between t_1 and t_2 is computed as $x = |t_1 - t_2| - 1$ (i.e. $t_1 \neq t_2$). The fuzzy time gap is defined as follows.

Definition 10. *Fuzzy Time Gap. Given two timestamps* t_1 *and* t_2*, a pair of one time gap* x *and one corresponding fuzzy set* a*, denoted by* $[x, a]$*, is called a fuzzy time gap if* $x = |t_1 - t_2| - 1$ *is involved in membership function* A*.*

For instance, see Figure 4.3, there are totally four time gaps which are $x_1 = 1, x_2 = 4, x_3 = 39$ and $x_4 = 948$. The fuzzy time gap $[x_2, weak]$, $[x_2, medium]$ and $[x_2, strong]$ respectively are $\mu_{weak}(x_2) = 0.8$, $\mu_{medium}(x_2) = 0.2$ and $\mu_{strong}(x_2) = 0$. Since x_4 is out of function A, it cannot be considered as a fuzzy time gap.

Definition 11. *Fuzzy Time Gap Set. Given an ordered list of timestamps* $T = \{t_{a_1}, t_{a_2}, \ldots, t_{a_m}\}$*, a set of time gaps* $X = \{x_1, \ldots, x_n\}, n = m - 1$*.* (X, A) *is a fuzzy time gap set generated from* T *if* $\forall i \in \{1, \ldots, n\} : x_i = |t_{a_i} - t_{a_{i+1}}| - 1$ *and* $\forall x \in X : x$ *is involved in* A*. Note that for any* $x \in X, x = 0$ *then* x *will be excluded from* X *without any affection.*

For instance, see Figure 4.3, a proper pattern $(\{o_1, o_2\}, \{t_2, t_4, t_5, t_{10}, t_{50}\})$ and a fuzzy time gap set is $X = \{x_1, x_2, x_3\}$ and for each time gap $x_i \in X$, there are a corresponding fuzzy set including $strong, medium$ and $weak$. Note that x_4 is out of membership function and therefore it is not included in X and $(\{o_1, o_2\}, \{t_2, t_4, t_5, t_{10}, t_{50}, t_{999}\})$ will not be considered as a valid pattern.

To highlight the participation of time gaps given by a fuzzy set a, we further propose an adaptation of the participation index [16] which is *fuzzy time gap participation index* proposed to take into account the fuzzy time gap occurrences in the pattern.

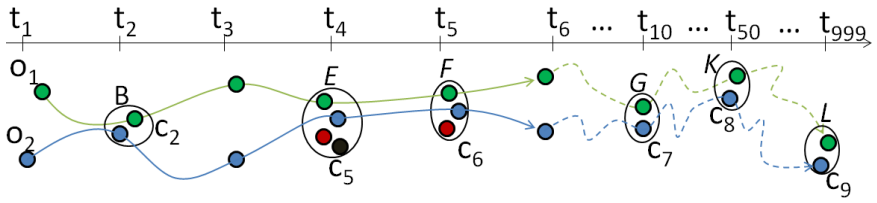


Figure 4.3: A fuzzy closed swarm example from our running example in Figure 1.1.

Definition 12. *Fuzzy Time Gap Participation Ratio. Let* (X, A) *be a set of fuzzy time gaps and* a *be an item of* A*, the fuzzy time gap participation ratio for* a *in* X *denoted* $\mathrm{TGr}(X, a)$ *can be defined as follows.*

$$\mathrm{TGr}(X, a) = \frac{\sum_{x \in X} \mu_a(x)}{|X|} \tag{4.1}$$

Definition 13. *Fuzzy Time Gap Participation Index. Let* (X, A) *be a set of fuzzy time gaps and* a *be an item of* A*, the fuzzy time gap participation index of* (X, A) *denoted* $\mathrm{TGi}(X)$ *can be defined as follows.*

$$\mathrm{TGi}(X) = \mathrm{Max}_{\forall a \in A} \mathrm{TGr}(X, a) \tag{4.2}$$

For instance, see Figure 4.3, a fuzzy time gap set $X = \{x_1, x_2, x_3\}$ and $\mathrm{TGr}(X, weak) = \frac{1+0.8+0}{3} = 0.6$, $\mathrm{TGr}(X, medium) = \frac{0+0.2+0.1}{3} = 0.1$, $\mathrm{TGr}(X, strong) = \frac{0+0+0.9}{3} = 0.3$. Thus, the fuzzy time gap participation index of X, $\mathrm{TGi}(X) = 0.6$.

Fuzzy swarm and fuzzy closed swarm. Given a group of objects O moving together in an ordered list of timestamps T and a set of fuzzy time gaps (X, A) generated from T. (O, T, X) is a fuzzy swarm that contains at least min_o objects (resp. $|O| \geq min_o$) during at least min_t timestamps (resp. $|T| \geq min_t$) and $\mathrm{TGi}(X) \geq \varepsilon$. The fuzzy swarm can be defined as follows.

Definition 14. *Fuzzy Swarm. Given integers* min_o, min_t *and a user-defined threshold* ε*.* (O, T, X) *is a fuzzy swarm if and only if:*

$$\begin{cases} (1) : |O| \geq min_o. \\ (2) : |T| \geq min_t. \\ (3) : (X, A) \text{ is a fuzzy time gap set.} \\ (4) : \forall i \in \{1, \ldots, n\}, \mathrm{TGi}(\{x_1, \ldots, x_i\}) \geq \varepsilon. \end{cases} \tag{4.3}$$

Note that if $X = \{x_1, x_2, x_3\}$ *then the condition (4) means that* $\mathrm{TGi}(\{x_1\}) \geq \varepsilon$, $\mathrm{TGi}(\{x_1, x_2\}) \geq \varepsilon$ *and* $\mathrm{TGi}(\{x_1, x_2, x_3\}) \geq \varepsilon$.

By definition, if we set $min_o = 2, min_t = 3$ and $\varepsilon = 0.2$ then there are totally 13 fuzzy swarms in Figure 4.3 such as $(\{o_1, o_2\}, \{t_2, t_4, t_5\}, \{x_1\})$, $(\{o_1, o_2\}, \{t_4, t_5, t_{10}\}, \{x_2\})$, $(\{o_1, o_2\}, \{t_2, t_5, t_{10}, t_{60}\}, \{x = 2, x_2\})$ and so on. However, it is obviously redundant to output fuzzy swarms like $(\{o_1, o_2\}, \{t_2, t_4, t_5, \})$ since it can be enlarged to $(\{o_1, o_2\}, \{t_2, t_4,$

$t_5, t_{10}, t_{50}\}, \{x_1, x_2, x_3\})$. To avoid mining redundant fuzzy swarms, we further give the definition of fuzzy closed swarm. Essentially, a fuzzy swarm (O, T, X) is *time-closed* if fixing T, O cannot be enlarged ($\nexists O'$ s.t. (O', T, X) is a fuzzy swarm and $O \subset O'$). Similarly, a fuzzy swarm (O, T, X) is *object-closed* if fixing O then T cannot be enlarged. Finally, a fuzzy swarm (O, T, X) is a fuzzy closed swarm if it is both *time-closed* and *object-closed*. Our goal is to find the complete set of fuzzy closed swarms. The definition is formally presented as follows.

Definition 15. *Fuzzy Closed Swarm. Given a fuzzy swarm* (O, T, X)*, it is a fuzzy closed swarm if and only if:*

$$\begin{cases} (1) : \nexists O', O \subset O' \wedge (O', T, X) \textit{ is a fuzzy swarm.} \\ (2) : \nexists T', T \subset T' \wedge (O, T', X') \textit{ is a fuzzy swarm.} \end{cases} \tag{4.4}$$

For instance (Figure 4.3), a closed swarm is $(\{o_1, o_2\}, \{t_2, t_4, t_5, t_{10}, t_{50}\}, \{x_1, x_2, x_3\})$.

Property 9. *Anti-monotonic. For all patterns* (O, T, X)*, if* (O, T, X) *is not a fuzzy swarm because of the condition (3) suffering then the following holds:*

For all supersets of (O, T, X) *by adding a later cluster and a fuzzy time gap in terms of time to* T *and* X *are not fuzzy swarms.*

Proof. After construction, we have $\exists k \in \{1, \ldots, n\}$ s.t. $TGi(\{x_1, \ldots, x_k\}) < \varepsilon$. For any $X' = \{x_1, \ldots, x_n, x_m\}$, (O, T', X') is not a fuzzy swarm since $\exists k \in \{1, \ldots, m\}$ s.t. $TGi(\{x_1, \ldots, x_k\}) < \varepsilon$. □

4.3 Discovering of Fuzzy Closed Swarms

The patterns we are interested in here, fuzzy closed swarms, is the association of a set of objects O, a set of timestamps T and a set of fuzzy time gaps X, denoted (O, T, X). As first glance, we can employ ObjectGrowth algorithm [23] to extract all closed swarms and then a post-processing step to obtain all the fuzzy closed swarms. However, moving object databases are naturally large and thus the search space of closed swarm extracting can be significantly increased (i.e. approximately $2^{|O_{db}|} \times 2^{|T_{db}|}$). Additionally, a huge amount of generated closed swarms (i.e. including extraneous patterns) can cause an expensive post-processing task. Furthermore, in real world applications (e.g. cars), object locations are continuously reported by using Global Positioning System (GPS). Thus, new data is always available and we need to execute again and again the algorithms on the whole database (i.e. including existing data and new data) to extract patterns. This is of course, cost-prohibitive and time consuming.

Table 4.1: Cluster matrix corresponding to our example in Figure 1.1.

T_{db}		t_1	t_2		t_3	t_4	t_5	t_{10}	t_{50}	t_{999}	
Clusters C_{db}		c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
O_{db}	o_1		1			1	1	1	1	1	
	o_2		1		1	1	1	1	1	1	
	o_3	1		1	1	1	1				1
	o_4	1		1	1	1					1

To deal with the issues, we propose *fCS-Miner* algorithm which is an adaptation of Incremental GeT_Move approach [12] [13] which has already been proved as being efficient in large moving object databases.

Basic idea of fCS-Miner algorithm. As in [12] [13], we first present C_{db} in a cluster matrix (see Table 4.1) so that Incremental GeT_Move can be applied to extract all frequent closed itemsets (FCIs). Next, we propose an novel property which can be used to directly extract fuzzy closed swarms from generated FCIs without a post-processing step. The cluster matrix definition is as follows.

Definition 16. *Cluster Matrix [12] [13]. Given a set of clusters* $C_{db} = \{C_1, C_2, \ldots, C_m\}$ *where* $C_i = \{c_{i_1 t_i}, c_{i_2 t_i}, \ldots, c_{i_n t_i}\}$ *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.

By applying Incremental GeT_Move which mainly bases on LCM algorithm [**?**] on the cluster matrix, we are able to extract all FCIs. Let us denote a frequent itemset as $\Upsilon = \{c_1, c_2, \ldots, c_k\}$, O_Υ contains the corresponding group of moving objects which are closed each other in a set of timestamps $T_\Upsilon = \{t(c_1), t(c_2), \ldots, t(c_k)\}$. We can recognize that $|O_\Upsilon| = \sigma(\Upsilon)$ [1], $|\Upsilon| = |T_\Upsilon|$ and X_Υ is used to denote as a fuzzy time gap set generated from T_Υ. For instance, see Table 4.1, a proper frequent itemset is $\Upsilon = \{c_2, c_5, c_6, c_7, c_8\}$ with $O_\Upsilon = \{o_1, o_2\}$, $T_\Upsilon = \{t_2, t_4, t_5, t_{10}, t_{50}\}$ and $X_\Upsilon = \{x_1, x_2, x_3\}$.

The following property, *f-closed swarm*, is used to verify whenever a frequent itemset Υ can be a fuzzy closed swarm or not.

Property 10. *f-Closed swarm. Given a frequent itemset* $\Upsilon = \{c_1, c_2, \ldots, c_k\}$, $X_\Upsilon =$

1. $\sigma(\Upsilon)$ is the support value of frequent itemset Υ.

$\{x_1,\ldots,x_n\}$. $(O_\gamma, T_\gamma, X_\gamma)$ *is a fuzzy closed swarm if and only if:*

$$\begin{cases} (1): \sigma(\Upsilon) \geq min_o. \\ (2): |\Upsilon| \geq min_t. \\ (3): \forall x \in X, x \textit{ is involved in } A. \\ (4): \forall i \in \{1,\ldots,n\}, TGi(\{x_1,\ldots,x_i\}) \geq \varepsilon. \\ (5): \nexists \Upsilon' \textit{ s.t } O_\gamma \subset O_{\gamma'}, T_{\gamma'} = T_\gamma \textit{ and } (O_{\gamma'}, T_\gamma, X_\gamma) \textit{ is a fuzzy swarm.} \\ (6): \nexists \Upsilon' \textit{ s.t. } O_{\gamma'} = O_\gamma, T_\gamma \subset T_{\gamma'} \textit{ and } (O_\gamma, T_{\gamma'}, X_{\gamma'}) \textit{ is a fuzzy swarm.} \end{cases} \tag{4.5}$$

Proof. After construction, we have $\sigma(\Upsilon) \geq min_o$ and thus $|O_\gamma| \geq min_o$ since $|O_\gamma| = \sigma(\Upsilon)$. Additionally, $|\Upsilon| \geq min_t$ and therefore $|T_\gamma| \geq min_t$ since $|\Upsilon| = |T_\gamma|$. Furthermore, $\forall x \in X : x$ is involved in A and $\forall i \in \{1,\ldots,n\}, TGi(\{x_1,\ldots,x_i\}) \geq \varepsilon$.. Consequently, $(O_\gamma, T_\gamma, X_\gamma)$ is a fuzzy swarm (*Definition 14*). Moreover, if $\nexists \Upsilon'$ s.t $O_\gamma \subset O_{\gamma'}, T_{\gamma'} = T_\gamma$ and $(O_{\gamma'}, T_\gamma, X_\gamma)$ is a fuzzy swarm then $(O_\gamma, T_\gamma, X_\gamma)$ cannot be enlarged in terms of objects. Therefore, it satisfies the *object-closed* condition. Furthermore, if $\nexists \Upsilon'$ s.t. $O_{\gamma'} = O_\gamma, T_\gamma \subset T_{\gamma'}$ and $(O_\gamma, T_{\gamma'}, X_{\gamma'})$ is a fuzzy swarm then $(O_\gamma, T_\gamma, X_\gamma)$ cannot be enlarged in terms of lifetime. Therefore, it satisfies the *time-closed* condition. Consequently, $(O_\gamma, T_\gamma, X_\gamma)$ is a fuzzy swarm and it satisfies *object-closed* and *time-closed* conditions and therefore $(O_\gamma, T_\gamma, X_\gamma)$ is a fuzzy closed swarm according to the *Definition 15*. □

To show the fact that from an itemset mining algorithm we are able to extract the set of all fuzzy closed swarms, we propose the following lemma.

Lemma 5. *Let* $FI = \{\Upsilon_1, \Upsilon_2, \ldots, \Upsilon_l\}$ *be the set of frequent itemsets being mined from the cluster matrix with* $minsup = min_o$. *All fuzzy closed swarms* (O, T, X) *can be extracted from* FI.

Proof. Let us assume that (O, T, X) is a fuzzy closed swarm. Note, $T = \{t(c_1), \ldots, t(c_k)\}$. According to the *Definition 15* we have $|O| \geq min_o$. If (O, T, X) is a fuzzy closed swarm then $\forall t(c_i) \in T, \exists c_i$ s.t. $O \subseteq c_i$ therefore $\bigcap_{i=1}^{k} c_i = O$. Additionally, we have $\forall c_i$, c_i is an item so $\exists \Upsilon = \bigcup_{i=1}^{k} c_i$ is an itemset and $O_\gamma = \bigcap_{i=1}^{k} c_i = O, T_\gamma = \bigcup_{i=1}^{k} t(c_i) = T$. Furthermore, we also have $X_\gamma = X$ as well. Therefore, $(O_\gamma, T_\gamma, X_\gamma)$ is a fuzzy closed swarm since $O_\gamma = O, T_\gamma = T$ and $X_\gamma = X$. So, (O, T, X) is extracted from Υ. Furthermore, $\sigma(\Upsilon) = |O_\gamma| = |O| \geq min_o$ then Υ is a frequent itemset and $\Upsilon \in FI$. Finally, $\forall (O, T)$ s.t. if (O, T, X) is a fuzzy closed swarm then $\exists \Upsilon$ s.t. $\Upsilon \in FI$ and (O, T, X) can be extracted from Υ, we can conclude that $\forall$ fuzzy closed swarm (O, T, X), it can be mined from FI. □

Essentially, by scanning the FCIs from the beginning to the end with the f-closed swarm property, we are able to extract the corresponding fuzzy closed swarms. The scanning process will be ended whenever one of conditions (3) and (4) in *Property 9* is suffered, after that the current frequent itemset Υ, i.e. $\sigma(\Upsilon) \geq min_o$, only need to be verified the conditions $|\Upsilon| \geq min_t$ and Υ contains the same number of objects with the FCI. This is because

Algorithm 5: fCS-Miner

```
   Input  : double ε, int min_o, int min_t, set of items C_db
1  begin
2  |  Incremental GeT_Move(C_db, min_o);
3  PatternMining(FCI, ε, min_t)
4  begin
5  |  f-CS := ∅;
6  |  if |FCI| ≥ min_t then
7  |  |  ϒ := ∅;
8  |  |  for k := 1 to |FCI| do
9  |  |  |  ϒ' := ϒ ∪ c_k;
10 |  |  |  if fuzzy(X_ϒ') = true ∧ TGi(ϒ') ≥ ε then
11 |  |  |  |  ϒ := ϒ';
12 |  |  |  else
13 |  |  |  |  if O_ϒ = O_FCI ∧ |ϒ| ≥ min_t + 1 then
14 |  |  |  |  |  f-CS := f-CS ∪ ϒ;
15 |  |  |  |  ϒ := ∅ ∪ c_k;
16 |  return f-CS;
17 where: fuzzy(X_ϒ') returns true if X_ϒ' is a fuzzy time gap set, otherwise returns false.
   In this function, we only need to verify that the last time gap is involved in A instead
   of all the time gaps in X_ϒ'.
```

Υ cannot be enlarged in terms of timestamps T_Υ (i.e. *Property 9*) and objects (i.e. FCI is closed). Thus, it satisfies all the requirements to be a fuzzy closed swarm completely.

The pseudo-code of fCS-Miner is presented in the Algorithm 5. We first apply Incremental GeT_Move on cluster matrix C_{db} with $minsup = min_o$ (line 2). Then, for each generated FCI, we directly scan it with the *f-closed swarm* property as mentioned before (lines 4-16). By using fCS-Miner, we are able to extract all fuzzy closed swarms on-the-fly without a post-processing step.

4.4 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 [23] [7] [12], the following dataset [3] have

2. http://www.lirmm.fr/~phan/fcsminer.jsp
3. http://www.movebank.org

been used during experiments: *Swainsoni dataset* includes 43 objects evolving over time and 764 different timestamps. It was generated from July 1995 to June 1998.

To the best of our knowledge, there is no previous work which addresses fuzzy closed swarms. Therefore, in the comparison, we employ the latest pattern mining algorithms such as CuTS* [4] [18] (convoy mining) and *ObjectGrowth* [23] (closed swarm mining). As pointed out in [23], *ObjectGrowth* outperforms *VG-Growth* [32] (group pattern mining) in terms of performance and therefore we will only consider *ObjectGrowth* and not both.

Similarly to [18] [23], we first use linear interpolation to fill in the missing data. Furthermore, as [18] [19] [23], DBScan [4] ($\mathtt{MinPts} = 2, \mathtt{Eps} = 0.001$) 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, to retrieve all the patterns including fuzzy closed swarms, convoys and closed swarms, in the reported experiments the fuzzy function in Figure 4.2 is applied, the default value of $\mathtt{min}_t$ is 1, $\mathtt{min}_o = 1$ and $\varepsilon = 0.001$. Note that the default values are the hardest conditions for examining all the algorithms.

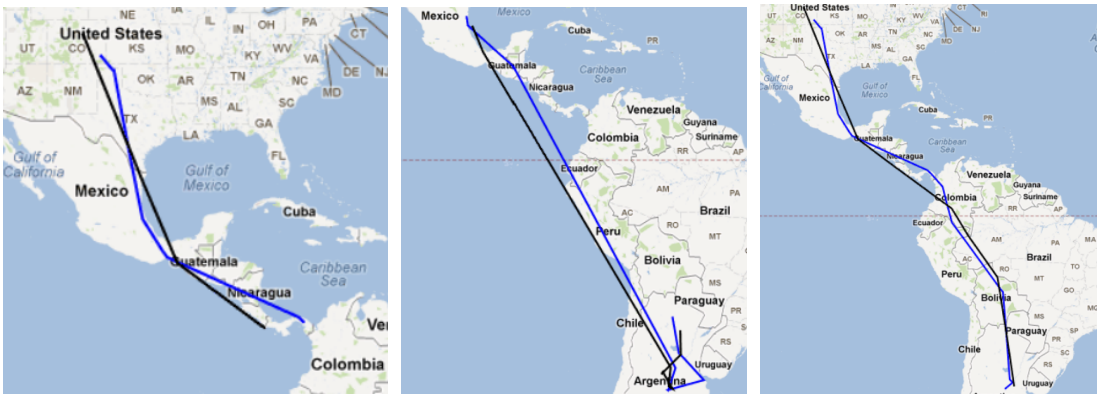
4.4.1 Effectiveness

The effectiveness of fuzzy closed swarms can be demonstrated through our online demo system. One of the extracted patterns from Swainsoni dataset is illustrated in Figure 4.4c. Each color represents a Swainsoni trajectory segment involved in the pattern.

To illustrate the feasibility of a fuzzy approach, we also show some of extracted closed swarms and convoys from our system [5] [8] in Figures 4.4a, b. We can consider that closed swarm is extraneous since the two objects meet each other at Mexico on October 1995 and after 5 months (i.e. to March 1996) for the next meeting location (i.e. Argentina). In fact, it is hard to say that they are moving together from Mexico to Argentina. While, the convoys are sensitive to time gaps and usually are short deal to the consecutive time constraint (see Figure 4.4a). Thus, they fail to describe the insightful relationship between objects. Either be too strict or too relaxed in dealing with time gaps may result in the loss of interesting patterns or reporting many uninteresting ones.

Distinguish from previous work, by proposing fuzzy closed swarms, we are able to reveal the relevant relationship between Swainsonies in a fuzzy point of view. Looking at the illustrated pattern in Figure 4.4c, we can consider that, from United States, the two objects are flying together along a narrow corridor through Central America and down to South America. Furthermore, they temporally diverge at Panama and congregate again at the Columbia central. The discovery of the fuzzy closed swarms on animal migration datasets provides useful information for biologists to better understand and examine the relationship and habits of these moving objects. Due to the space limitation, we do not provide

4. The source code of CuTS* is available at http://lsirpeople.epfl.ch/jeung/source_codes.htm
5. http://www.lirmm.fr/~phan/index.jsp



(a) A convoy. (b) A closed swarm. (c) A fuzzy closed swarm.

Figure 4.4: An example of extracted patterns from Swainsoni dataset. The two object names are 'SW22' and 'SW40'.

experiments by varying the fuzzy membership function A. However, in real world context, users can express their expertise through the membership function for dealing with fuzzy approximate reasoning issues.

4.4.2 Parameter Sensitiveness

To show the parameter sensitiveness and efficiency of the proposed algorithm, as in [23], we also generate a large synthetic dataset using Brinkhoff's network [6]-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 with slow moving speed (5×10^6 points in total). DBScan ($MinPts = 3, Eps = 300$) is applied to obtain clusters.

Sensitiveness w.r.t ε. See Figure 4.5a, we can consider that fCS-Miner is not sensitive in ε. This is because ε is only used to scan the FCIs for fuzzy closed swarm extraction which is much less expensive than FCI mining task.

Sensitiveness w.r.t min_t. Figure 4.5b 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 (Figure 4.6b). Meanwhile, fCS-Miner and CuTS* only use min_t at the pattern reporting or verifying steps without any pruning rule for min_t. Additionally, as mentioned before the fuzzy closed swarm verifying task is less expensive than the FCI extraction. Consequently, be similar to CuTS*, the fCS-Miner sensitiveness in min_t is less sensitive than ObjectGrowth.

6. http://iapg.jade-hs.de/personen/brinkhoff/generator/

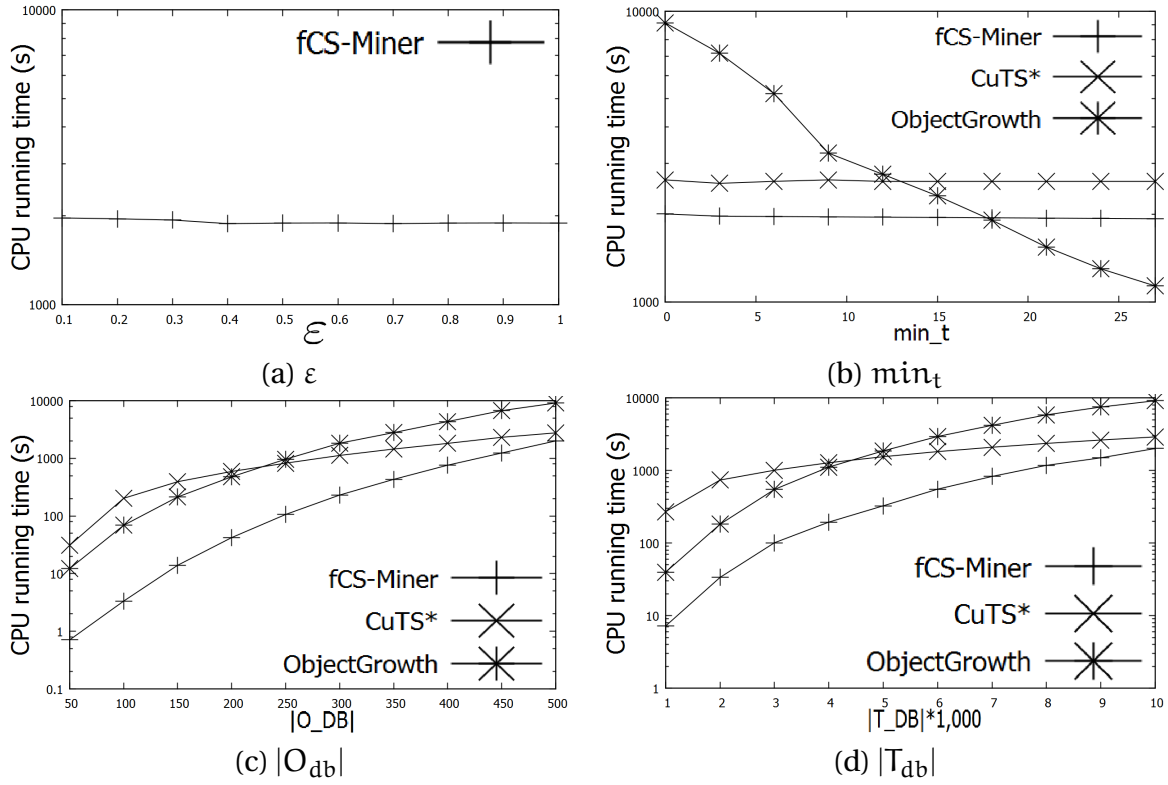


Figure 4.5: Running time on Synthetic Dataset.

Sensitiveness w.r.t O_{db}, T_{db}. Figures 4.5c-d show the sensitiveness in the sizes of O_{db} and T_{db}. We can consider that all the algorithms are quite similar to each other. However, CuTS* is a little bit less sensitive than the others. This is because, in CuTS*: 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. While, for ObjectGrowth, 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 (see Figures 4.6c-d). This behavior is similar in fCS-Miner since the number of FCIs can be large. However, thanks to the fuzzy approach, there are not huge amount of generated patterns compared to ObjectGrowth. Obviously, fCS-Miner is similar to ObjectGrowth and a little bit more sensitive than CuTS* in terms of $|O_{db}|$ and $|T_{db}|$.

Influence of $TGi(X)$ on #f-Closed swarms. Figure 4.7 shows the influence of the fuzzy time gap participation index on the number of patterns that contain weak, medium and strong time gaps. We can consider that the number of patterns which have $TGi(X)$ with weak fuzzy time gaps X, medium fuzzy time gaps X and strong fuzzy time gaps X are quite distinguished from each other. Since, the number of patterns with weak X is smallest, more

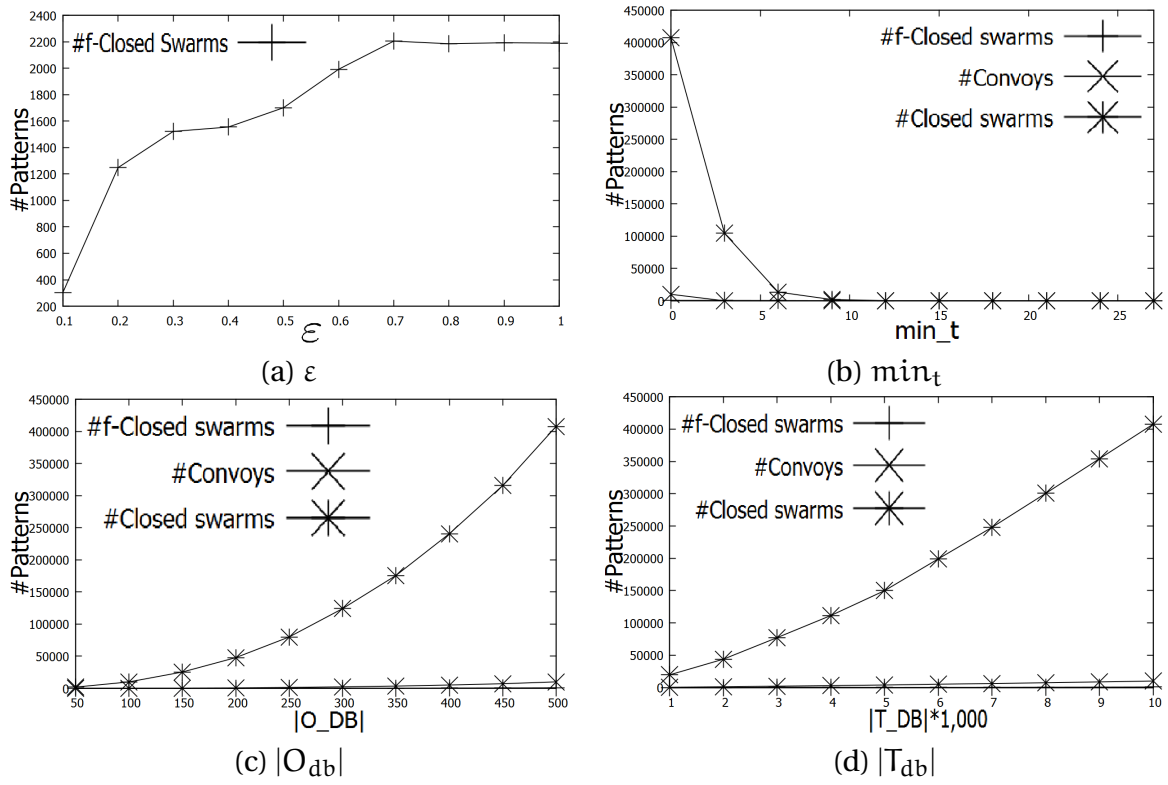


Figure 4.6: Number of patterns on Synthetic Dataset.

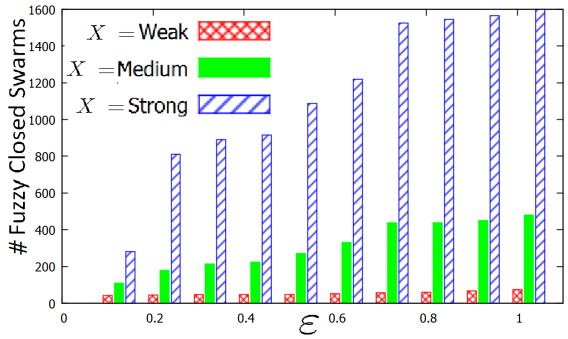


Figure 4.7: Influence of TGi(X) on #patterns through ε.

number of patterns with medium X and the highest number of patterns with strong X. Therefore, the TGi(X) enable us to rank the fuzzy closed swarms well corresponding with the membership degree function. Furthermore, if we ignore all the fuzzy closed swarms with strong (and medium) fuzzy time gaps, a number of uninteresting patterns can be eliminated.

To summarize, fCS-Miner is effective to extract fuzzy closed swarms which are novel

and useful movement patterns. By applying fuzzy function, users can express their background knowledge in order to obtain interesting patterns without generating extraneous ones. Additionally, fCS-Miner parameter sensitiveness is quite acceptable compare to the other model algorithms. Moreover, with the purpose to extract the complete set of f-closed swarms, fCS-Miner is competitive in time efficiency to state-of-the-art approaches (see Figure 4.5).

4.5 Discussion

In this chapter, to deal with the issue that enforcing the consecutive time constraint or completely relaxing may result in the loss of interesting patterns or the generation of uninteresting patterns, we propose the concept of fuzzy swarm which softens the time gap constraint. These concepts enable the discovery of insightful movement patterns and the elimination of extraneous patterns. A new method fCS-Miner is proposed to efficiently extract all the fuzzy closed swarms. The proposed algorithm's effectiveness, and parameter sensitiveness are demonstrated using real and large synthetic datasets. In the near future work, the proposed approaches can be applied on other kinds of patterns (e.g. gradual trajectory patterns [10]).

Although the fuzzy-closed swarm and existing movement patterns are very meaningful, they cannot help us to capture the object moving trends. To illustrate, let us consider the Salmon [7] migration in the ocean, where the adult salmons return primarily to their natal stream to spawn. Therefore, from time to time, more and more salmons get closed together to go to their stream origin. Obviously, by focusing only on an unchanged group of moving objects, traditional approaches are not able to capture such kind of moving trends. Actually, this phenomenon is involved in many real world applications such as traffic congestion, animal or population migration, etc.

For instance, the pattern: *"from June to July, as time passes, the more people are going to Miami"* is useful for service providers such as travel agencies, hotels, restaurants to prepare reasonable business strategies. The extracted knowledge from the pattern *"from October to December, as time passes, the more Eagles are moving from Canada to Mexico"* is especially useful for biologists to plan conservation activities such as migration analysis, counting, telemetry attaching, etc. Thus, what we can show is that in many scenarios, extracting object moving trends is interesting and useful.

This issue is addressed in the next chapter where we present the *time relaxed gradual moving object cluster* concept to express the object moving trends.

7. http://en.wikipedia.org/wiki/Salmon

CHAPTER

5 Mining Time Relaxed Gradual Moving Object Clusters

Preamble

Traditionally, existing movement pattern mining 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 contexts. In this chapter, we present a novel concept of "time relaxed gradual trajectory pattern", denoted rGpattern, which captures the object movement tendency. Moreover, this chapter proposes an algorithm, named CLUSTERGROWTH*, for the discovery of all interesting maximal rGpatterns, and three pruning strategies to reduce the search space.*

In order to enrich the utility of the rGpattern concept, we adapt the Minimum Description Length (MDL) principle for mining representative rGpatterns. An encoding scheme which is designed to deal with different kinds of overlapping rGpattern structures is proposed. We show that mining representative rGpatterns is NP-Hard and therefore we propose two heuristic algorithms to extract compressing rGpatterns. The first algorithm, named COMPOGP*, uses a greedy two-phase approach. To overcome performance with the required candidate generation in* COMPOGP *we propose an effective algorithm, called* DICOMPOGP*, to directly mine compressing rGpatterns. Conducted experiments on real and large synthetic datasets demonstrate the pattern meaning, effectiveness and efficiency of our proposed methods.*

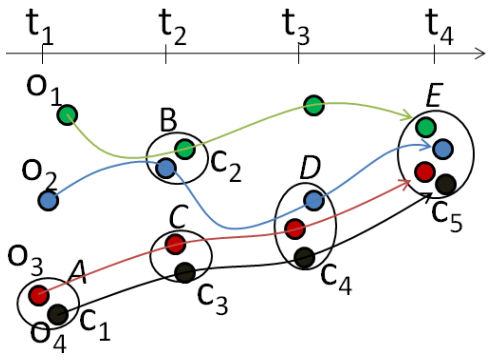


Figure 5.1: An example of gradual moving object clusters from running example in Figure 1.1.

5.1 Introduction

As exemplified in Figure 5.1, the retrieved knowledge from traditional patterns can be *"the two cars,* o_1 *and* o_2*, are moving together from* t_2 *to* t_4*"* or *"the two cars,* o_3 *and* o_4*, are moving together from* t_1 *to* t_4*"*. Even if these patterns are interesting, they do not really present the actual moving behavior which can be *"from* t_1 *to* t_4*, as time passes, the more objects are following the trajectory* $\{A \rangle C \rangle D \rangle E\}$*"*.

In this chaper, we propose a novel movement pattern, *gradual moving object cluster*, which is designed to capture the gradual object moving trend. More precisely, a gradual moving object cluster is a list of moving object clusters which satisfy the graduality constraint and integrity condition during at least min_t consecutive 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. We will clearly define them in the next sections.

For instance, see Figure 5.1, given $min_t = 2$, a list of clusters $C = \{c_1, c_3, c_4, c_5\}$ can be considered as a gradual moving object cluster. Since, the cluster sizes are continuously increasing and the object in a cluster at time t is still grouped with the other ones at time $t+1$.

Naturally, moving objects in a cluster may actually diverge temporarily and congregate at certain timestamps. Therefore, the requirement that a gradual moving object cluster to be at least min_t consecutive timestamps, which might not be practical in the real cases. For instance, if we set $min_t = 3$ in Figure 5.2, no gradual moving object cluster can be found. However, intuitively, these object moving trend is *"from* t_1 *to* t_6*, the more time passes, the more objects are following the trajectory* $\{A \rangle B \rangle D \rangle F\}$*"* even though they temporarily diverge at some snapshots. If we relax the consecutive time constraint and still set $min_t = 3$, actually we can detect these object moving trend. In other words, enforcing the consecutive time constraint may result in the loss of interesting patterns.

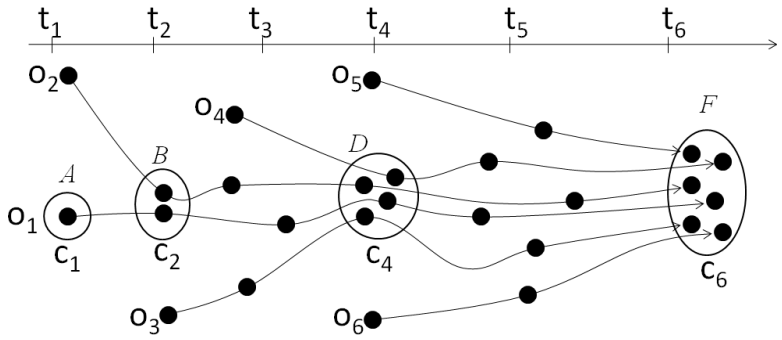


Figure 5.2: An example of time relaxed gradual trajectory pattern.

To tackle this issue, we propose the concept of *rGpattern* (i.e. time relaxed gradual trajectory pattern) which is a more general type of gradual moving object clusters. If we denote a rGpattern as a list of clusters C then, in Figure 5.2, we have four rGpatterns: $C_1 = \{c_1, c_2, c_4\}$, $C_2 = \{c_1, c_2, c_6\}$, $C_3 = \{c_2, c_4, c_6\}$ and $C_4 = \{c_1, c_2, c_4, c_6\}$. We can note that they are redundant since C_1, C_2, C_3 are included in C_4. To avoid finding redundant rGpatterns, we further propose the *maximal rGpattern* concept. The basic idea is that if C is a rGpattern, it is unnecessary to output any subset C' of C even if C' may also satisfy rGpattern requirements. For instance, see Figure 5.2, a maximal rGpattern is $C_4 = \{c_1, c_2, c_4, c_6\}$ while C_1, C_2 and C_3 are not maximal.

Efficient extracting of complete set of maximal rGpatterns in a large moving object database, denoted db, is a non-trivial task. First, the size of all the possible combinations is exponential (i.e. approximately $2^{|C_{db}|}$) where $|C_{db}|$ is significantly larger than $|T_{db}|, |O_{db}|$ (i.e. $|C_{db}| > |T_{db}| \times |O_{db}|$ in some cases). Second, none of previous work (i.e. frequent pattern mining [1] [15], moving object clusters [23] [18] [32] [17] [19]) solves exactly the same issue as finding maximal rGpatterns. This is because they do not address the graduality in terms of itemsets or moving clusters. However, graduality is a key feature in rGpattern context. Thus, the discovery of rGpatterns introduces a new problem that needs to be solved by specifically designed techniques.

Facing the huge potential search space, we propose an efficient approach, named *ClusterGrowth*. The approach shares the same spirit with the ObjectGrowth algorithm [23] but be different in terms of design and goal. In ClusterGrowth, we design two efficient rules which are *Graduality Pruning rule* and *Backward Pruning rule* to end unnecessary further search. Additionally, to eliminate uninteresting patterns, we relax the time constraint within a time-based sliding window. After the pruning rules cut a great portion of invalid candidates, we also present an *Actual Maximum Checking* step that reports the *interesting maximal rGpatterns* on-the-fly without extra space to store candidates and extra time for post-processing.

In order to enrich the utility of the rGpattern concept we adopt the Minimum Descrip-

tion Length (MDL) principle for mining representative rGpatterns. An encoding scheme which is designed to deal with different kinds of overlapping rGpattern structures is proposed. We show that mining representative rGpatterns is NP-Hard and therefore we propose two heuristic algorithms to extract compressing rGpatterns. The first algorithm, named CompoGp, uses a greedy two-phase approach. To overcome performance with the required candidate generation in CompoGp we propose an effective algorithm, called DiCompoGp, to directly mine compressing rGpatterns.

In summary, the main contributions of this chapter are as follows.

1. We first propose the concept of rGpattern, i.e. time relaxed gradual moving object cluster, which captures the object moving trend. To discover the complete set of rGpatterns, we propose the ClusterGrowth approach in which three efficient pruning rules are designed to reduce the search space.
2. We prove that mining top-K compression rGpatterns based on MDL principle [6] is NP-Hard problem.
3. We propose a two-phase algorithm, named CompoGp, for mining representative rGpatterns inspired by the original Krimp algorithm [31].
4. We propose DiCompoGp, an efficient algorithm which avoids the expensive candidate generation phase and directly mines representative rGpatterns from the data.
5. Experimental results on real-life and synthetic datasets demonstrate the effectiveness and efficiency of the proposed approaches.

The remaining sections of the chapter are organized as follows. The definitions of rGpatterns and (interesting) maximal rGpatterns are given in Section 2. ClusterGrowth will be clearly presented in Section 3. Experiments testing effectiveness, parameter sensitiveness and efficiency are shown in Section 4. Mining representative gradual trajectory pattern issue will be presented in Section 5 and experimental results in Section 6. Finally, we draw our conclusions in Section 7.

5.2 Problem Statement

In this section, we give the rGpattern and maximal rGpattern 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 $* \in \{\geq, \leq\}$ (see Table 5.1).

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

Definition 17. *rGpattern. Given a list of clusters* $C^* = \{c_1, \ldots, c_n\}$ *and a minimum threshold*

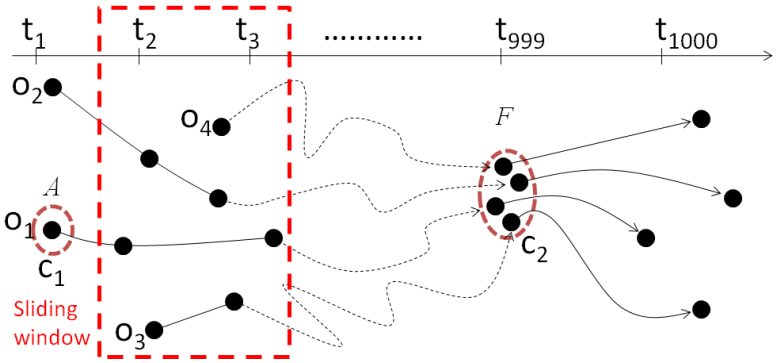


Figure 5.3: An example of uninteresting rGpattern and sliding window ($w = 2$).

$\min_t$. C^* *is a rGpattern if:*

$$C^* = C^{\geq}: \begin{cases} (1): |C^*| \geq \min_t. \\ \forall i \in \{1,\ldots,n-1\}, \\ (2): c_i \subseteq c_{i+1}. \\ (3): |c_n| > |c_1|. \end{cases} \tag{5.1}$$

$$C^* = C^{\leq}: \begin{cases} (1): |C^*| \geq \min_t. \\ \forall i \in \{1,\ldots,n-1\}, \\ (2): c_i \supseteq c_{i+1}. \\ (3): |c_n| < |c_1|. \end{cases} \tag{5.2}$$

For instance, see Figure 5.2, 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 rGpattern since $|C_1^{\geq}| \geq \min_t$, $c_1 \subset c_2 \subset c_4$ and $|c_4| = 4 > |c_1| = 1$. Furthermore, there are totally 4 rGpatterns: $C_1^{\geq} = \{c_1, c_2, c_4\}$, $C_2^{\geq} = \{c_1, c_2, c_6\}$, $C_3^{\geq} = \{c_2, c_4, c_6\}$ and $C_4^{\geq} = \{c_1, c_2, c_4, c_6\}$.

However, it is obviously redundant to output rGpatterns like $C_1^{\geq}, C_2^{\geq}, C_3^{\geq}$ since all of them can be enlarged to $C_4^{\geq}$. Naturally, it is necessary to ignore the redundant patterns. Therefore, we only focus on extracting maximal rGpatterns.

Definition 18. *Maximal rGpattern. Given a rGpattern* $C^* = \{c_1,\ldots,c_n\}$. C^* *is maximal if* $\nexists C'^*, C^* \subset C'^*$ *and* C'^* *is a rGpattern.*

For instance, in Figure 5.2, $C_4^{\geq} = \{c_1, c_2, c_4, c_6\}$ is a maximal rGpattern.

Uninteresting rGpatterns. As mentioned before, 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 5.3, $\{c_1, c_2\}$ is a rGpattern but it is not interesting and useless. Since the objects o_1, o_2, o_3, o_4 do not essentially move together during the time following the trajectory *{A ⟩ F}*, and they only meet each other at F by chance after 999 timestamps. To

Table 5.1: Notation Description.

Notation	Description
$O_{db} = \{o_1, o_2, \ldots, o_z\}$	Object set
$T_{db} = \{t_1, t_2, \ldots, t_m\}$	Set of timestamps
C_{db}	Set of clusters extracted from a dataset
$\lvert O_{db} \rvert$	The size of O_{db}
$\lvert T_{db} \rvert$	The size of T_{db}
$\lvert C_{db} \rvert$	The size of C_{db}
$* \in \{\geq, \leq\}$	Gradual variation
$C^* = \{c_1, \ldots, c_n\}$	Interesting (maximal) rGpattern w.r.t $*$
$T_{C^*} = \{t(c_1), \ldots, t(c_n)\}$	List of timestamps w.r.t C^*
min_t	A minimum #timestamps within that objects are grouped together
w	(Time-based) sliding window size
oK	Object key

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 (see Figure 5.3). 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 5.3, given that $w = 2$, $\{c_1, c_2\}$ is a uninteresting pattern since $t(c_2) - t(c_1) = 998 > w = 2$. The interesting pattern definition is as follows.

Definition 19. *Interesting Maximal rGpattern. Given a maximal rGpattern* $C^* = \{c_1, \ldots, c_n\}$*, a time-based sliding window size* w*.* C^* *is an interesting pattern if:*

$$\forall i \in \{1, \ldots, n-1\} : 1 \leq t(c_{i+1}) - t(c_i) \leq w \tag{5.3}$$

For conciseness sake, the term *"sliding window"* will be used to indicate the *"time-based sliding window"* in the rest of this chapter.

5.3 Discovering Maximal Time Relaxed Gradual Trajectory Patterns

The patterns we are interested in here, interesting maximal rGpattern, is a list of clusters C^*. At first glance, the number of different rGpatterns could be $2^{|C_{db}|}$ (i.e. the size of

Table 5.2: An example of a reconfigured spatio-temporal database in Figure 5.1.

Timestamp	Object Key	Cluster
t_1	0011	c_1
t_2	1100	c_2
t_2	0011	c_3
t_3	0111	c_4
t_4	1111	c_5

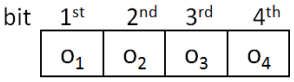


Figure 5.4: An object bit set configuration example.

the search space). Second, we need to store the object information for each cluster and then compute the intersections between them and that is, we have to be careful in concerning the structures that we use. From the analysis above we propose to reconfigure spatio-temporal databases so that for each timestamp and for each cluster, a bit set presentation, called object key, is used to illustrate the object information (i.e. see Table 5.2, Figure 5.4). For clarity sake, we take the database reconfiguration as a preprocessing step.

Basic idea of ClusterGrowth algorithm. For the search space of C_{db}, we apply a depth-first search of all subsets of C_{db}, which is illustrated as a pre-order tree traversal in Figure 5.5: 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*).*

Even though, the search space is huge, efficient pruning rules are demanding to speed up the search process. We propose two pruning rules to further shrink the search space. The former, called *Graduality Pruning*, is to end traversing the subtree when we find further traversal cannot satisfy graduality and interestingness requirement (Definition 37). The latter, called *Backward Pruning*, is to make use of 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. Armed with these two pruning rules, the size of the search space can be significantly reduced.

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

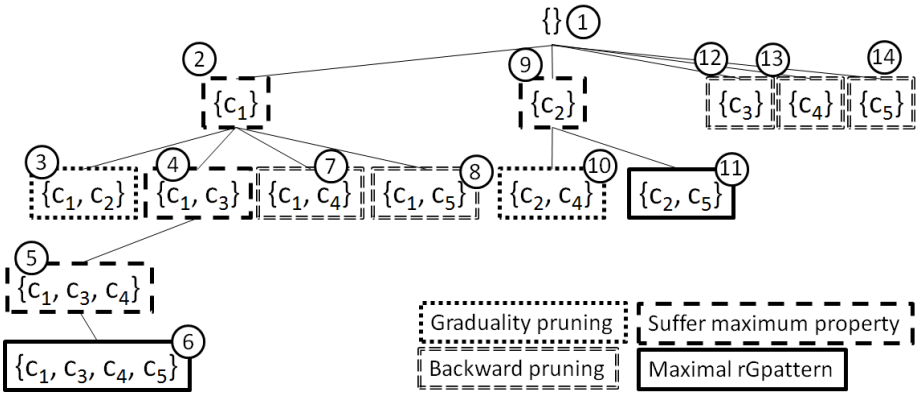


Figure 5.5: ClusterGrowth search space of the running example in Figure 5.1 with $* = \text{'}\geq\text{'}$, $\min_t = 1$ and $w = 3$.

5.3.1 ClusterGrowth Approach

In this section, we will clearly define our proposed pruning rules. The ClusterGrowth method is a depth-first-search (DFS) approach based on the cluster set search space.

Property 11. ***(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 rGpattern if:*

$$C^* = C^{\geq} : (t(c') - t(c_n) > w) \vee (c' \not\geq c_n) \tag{5.4}$$

$$C^* = C^{\leq} : (t(c') - t(c_n) > w) \vee (c_n \not\geq c') \tag{5.5}$$

Proof. After construction, we have $(C^* \cup c')$ is not an interesting maximal rGpattern since c_n and c' do not satisfy Definitions 17 and/or 37. Consequently, for any superset C'^* of $(C^* \cup c')$, C'^* is not an interesting maximal rGpattern. □

In Figure 5.5, the nodes with list of clusters $C^{\geq} = \{c_1, c_2\}$ and its subtree are pruned by Graduality Pruning rule because $c_1 \not\leq c_2$. This is similar for $\{c_2, c_4\}$.

Even though Graduality Pruning rule can eliminate a large number of useless candidates, there are many other candidates need to be pruned to shrink the search space to the optimal point. For instance, $\{c_3\}$ subtrees cannot provide any interesting maximal rGpatterns. This is because $\{c_1, c_3\}$ has been already traversed and $\{c_1, c_3\}$ is a rGpattern. Therefore, for any superset of $\{c_3\}$, denoted $\{c_3\} \cup C^*$, if it is a rGpattern then we also have $\{c_1, c_3\} \cup C^*$ is a rGpattern. Thus, $\{c_3\} \cup C^*$ is not maximal and therefore $\{c_3\}$ subtrees need to be pruned. The *Backward Pruning* rule can be formalized as follows.

Property 12. ***(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 17 (i.e. graduality condition) then any supersets of C^* are not interesting maximal rGpatterns. Thus, C^* subtrees need to be pruned.

Proof. After construction, we have $\{c_1,\ldots,c',\ldots,c_n\}$ satisfies the graduality condition (i.e. if $*=$ '$\geq$' then $c_1 \subseteq \ldots \subseteq c' \subseteq \ldots \subseteq c_n$, or if $*=$ '$\leq$' then $c_1 \supseteq \ldots \supseteq c' \supseteq \ldots \supseteq c_n$). For any supersets C'^* of C^*, $C'^* = C^* \cup C''^*$, if C'^* is a rGpattern then $\{c_1,\ldots,c',\ldots,c_n\} \cup C''^*$ is a rGpattern. Since $C'^* \subset \{c_1,\ldots,c',\ldots,c_n\} \cup C''^*$, C'^* is not an interesting maximal rGpattern because it suffers the Definition 18. Thus, C^* subtrees need to be pruned. □

Backward Pruning is efficient in the context that it only needs to examine those supersets of C^* with one more cluster rather than all the supersets. By applying this rule, we can efficiently prune a significant portion of the search space for mining interesting maximal rGpatterns.

Actual maximum checking. After pruning all useless candidates, we need to verify the remaining ones for obtaining the complete set of interesting maximal rGpatterns. 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 17.

Property 13. ***(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 17 then C^* is not an interesting maximal rGpattern.*

Proof. After construction, we have $C^* \subset C'^*$. Additionally, if C^* is a rGpattern then C'^* is also a rGpattern. Therefore C^* is not maximal because it suffers the Definition 18. □

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

Theorem 1. ***(Interesting maximal rGpattern in ClusterGrowth)***. *Given a node with list of clusters $C^* = \{c_1,\ldots,c_n\}$, C^* is an interesting maximal rGpattern 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 17 (i.e. if $*=$ '$\geq$' then $|c_n| > |c_1|$, or if $*=$ '$\leq$' then $|c_n| < |c_1|$).*

Proof. For every list of clusters C^* passes Graduality Pruning, Backward Pruning and Actual Maximum rule, first of all, Graduality Pruning ensures that C^* satisfies the condition 2-Definition 17 (i.e. graduality condition) and the Definition 37 (i.e. interestingness condition). Furthermore, Backward Pruning and Actual Maximum rules ensure that C^* satisfies the maximum property (i.e. Definition 18). If $|C^*| \geq \min_t$ and c_1, c_n satisfy the condition

3-Definition 17 (i.e. $|c_n| > |c_1|$ where $C^* = C^{\geq}$ or $|c_n| < |c_1|$ where $C^* = C^{\leq}$) then $|C^*|$ completely satisfies the Definition 17. Thus, C^* satisfies all the requirements (Definitions 17, 18, 37) to be an interesting maximal rGpattern. □

Theorem 1 makes the discovery of interesting maximal rGpatterns well embedded in the search process so that interesting maximal rGpatterns can be reported from the algorithm directly without a post-processing step.

Figure 5.5 shows the complete ClusterGrowth algorithm for our running example in Figure 5.1. We start traversing the search space at $\{\}$ and then visit the node $\{c_1\}$ which passes the Backward Pruning since it is the first cluster in C_{db}. Next, we visit the node $\{c_1, c_2\}$ and it suffers Graduality Pruning rule since $c_1 \not\subseteq c_2$. Next, we visit the node $\{c_1, c_3\}$ and it satisfies Backward Pruning rule as well as Graduality Pruning rule since $0 < t(c_3) - t(c_1) < w$ and $c_1 \subset c_2$. By applying Actual Maximum property $\{c_1\}$ is not maximal. Then, by following its subtrees we have that $\{c_1, c_3, c_4, c_5\}$ passes both Backward Pruning and Graduality Pruning. Furthermore, $|\{c_1, c_3, c_4, c_5\}| \geq min_t$ and $|c_5| > |c_1|$, and thus $\{c_1, c_3, c_4, c_5\}$ is an interesting maximal rGpattern (Theorem 1). Now, we trace back and visit the nodes $\{c_1, c_4\}$ and $\{c_1, c_5\}$. Indeed, they are not maximal rGpatterns since they suffer the Backward Pruning rule. We continue visiting the nodes until we finish the traversal of the cluster set-based DFS tree.

5.3.2 The ClusterGrowth Implementation

An object key is a list of $|O_{db}|$ binary bits (see Figure 5.4) and let us define a set of operations that will be used for interesting maximal rGpattern extracting in the ClusterGrowth algorithm. Note that & is used to denote the bitwise operation *AND*.

– **Contain**(oK_1, oK_2): given two object keys oK_1 and oK_2, returns true if oK_1 & $oK_2 = oK_2$.
– $|oK|$: given an object key oK, returns the number of '1's in oK.
– $c \subseteq c'$: given two clusters c and c', returns **Contain**($oK(c')$, $oK(c)$).
– $c \supseteq c'$: given two clusters c and c', returns **Contain**($oK(c)$, $oK(c')$).
– $|c|$: given a cluster c, returns $|oK(c)|$.

For instance, see Table 5.2, at timestamp t_1, we have three clusters $c_1 = \{o_3, o_4\}, c_2 = \{o_1, o_2\}, c_3 = \{o_3, o_4\}$. Thus, $oK(c_1) =$ '0011', $oK(c_2) =$ '1100' and $oK(c_3) =$ '0011'.

Algorithm 6 presents the pseudo code for the ClusterGrowth algorithm. The algorithm takes as inputs a preprocessed database and parameters min_t, w and $*$. We first create two variables op_1 and op_2 to represent the gradual variation $*$ (lines 2-5). op_1 and op_2 are respectively used for set operations and cluster size comparisons between 2 clusters. Then, we start the process with each cluster $c \in C_{db}$ by calling the function $ClusterGrowth(C^*, op_1, op_2, c, min_t, w)$.

When traversing the node with list of clusters C^*, we first check whether a cluster c which is in the sliding window w can be added to C^* by passing Graduality Pruning rule

Algorithm 6: Interesting Maximal rGpattern Mining Algorithm

Input : A database (O_{db}, T_{db}), a cluster database C_{db}, int min_t, int w, $*$
Output: all interesting maximal rGpatterns

begin
  if $* := '\geq'$ **then**
    $op_1 = \subseteq; op_2 := \geq;$
  else
    $op_1 = \supseteq; op_2 := \leq;$
  $realGpatterns := \emptyset;$
  foreach $c \in C_{db}$ **do**
    $C^* := \emptyset;$
    $C^* := C^* \cup c;$
    $ClusterGrowth(C^*, op_1, op_2, c, min_t, w);$
  return $realGpatterns;$
ClusterGrowth$(C^*, op_1, op_2, c\_last, min_t, w)$
begin
  if $Backward(C^*, op_1, c\_last) = false$ **then**
    return;
  $maximum := true;$
  foreach $c \in C_{db}$ *s.t.* $t(c) - t(c\_last) \leq w$ **do**
    if $c\_last\ op_1\ c$ **then**
      $maximum := false;$
      $C'^* := C^* \cup c;$
      $ClusterGrowth(C'^*, op_1, op_2, c, min_t, w);$
  if $maximum = true \wedge |C^*| \geq min_t \wedge |c\_last|\ op_2\ |c\_first|$ **then**
    $realGpatterns := realGpatterns \cup C^*;$
Backward(C^*, op_1, c_last)
begin
  foreach $c' \notin C^*, c' < c\_last$ **do**
    if $c_i\ op_1\ c'\ op_1\ c_j$ **then**
      return false;
  return true;

where: c_last, c_first respectively are c_n, c_1 in C^*, in Backward function, c_i, c_j are two clusters in C^* and they are nearest clusters to c' so that $c_i < c' < c_j$. Note that, to extract all interesting maximal rGpatterns $C^{\geq}$ and $C^{\leq}$, we apply the algorithm with different values of $*$ (i.e. $* = '\geq'$ and $* = '\leq'$).

(lines 17-18). If so, we note that C^* cannot pass the Actual Maximum rule by assigning false to $maximum$ (line 17). After that, we generate a superset of C^*, denoted C'^*, by adding c into C^*, then start the new search process with C'^* (lines 18-19).

Next, we check whether C^* can pass the Backward Pruning (lines 14-15). In Backward function (lines 24-29), we only need to check whether c_i, c' and c_j satisfy the graduality constraint instead of the entire list of clusters C^*. For instance, if $C^* = \{c_1, c_2, c_6, c_7\}$ and $c' = c_4$ then c_i and c_j respectively are c_2 and c_6. We only need to check that c_2, c_4, c_6 satisfy the graduality constraint or not. If so, the current list of clusters C^* cannot pass the Backward Pruning.

Finally, we check whether the current list of clusters C^* can satisfy the Actual Maximum rule, min_t constraint and condition 3-Definition 17 with c_last, c_first (resp. c_n, c_1). If so, C^* will be imported into $realGpatterns$ (lines 20-21) for reporting later (line 11).

5.4 Preliminarily Experimental Results

Before we continue with more theory, we will first present some results on pattern meaning and efficiency of the ClusterGrowth algorithm. A 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 [1]. As in [8], the two following datasets [2] have been used during experiments: *Swainsoni dataset* includes 43 objects evolving over time and 764 different timestamps. It was generated from July 1995 to June 1998. *Buffalo dataset* concerns 165 buffaloes with the tracking time from year 2000 to year 2006. The original data has 26,610 reported locations and 3001 timestamps.

To the best of our knowledge, there is no previous work which addresses maximal rG-pattern. Therefore, in the comparison, we employ the latest pattern mining algorithms such as CuTS* [3] [18] (convoy mining) and *ObjectGrowth* [23] (closed swarm mining). As pointed out in [23], *ObjectGrowth* outperforms *VG-Growth* [32] (group pattern mining) in terms of performance and therefore we will only consider *ObjectGrowth* and not both.

Similarly to [23] [18], we first use linear interpolation to fill in the missing data. Furthermore, as [23] [18] [12], DBScan [4] ($MinPts = 2, Eps = 0.001$) 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, to retrieve all the patterns including interesting maximal rGpatterns, convoys and closed swarms, in the reported experiments, the default value of min_t is 1,

1. http://www.lirmm.fr/~phan/realgp.jsp
2. http://www.movebank.org
3. The source code of CuTS* is available at http://lsirpeople.epfl.ch/jeung/source_codes.htm

$\mathtt{min}_o = 1$ (i.e. for ObjectGrowth and CuTS*). Note that the default values are the hardest conditions for examining all the algorithms. Regarding to the sliding window size, the window size can be varied depend on different applications such as cars along highways closing each other in every an hour, or people come to same places every day, etc. In this chapter, w is set to $10\%|T_{db}|$ for each dataset. It means that $w = 76$ days for Swainsoni dataset, $w = 300$ days for Buffalo dataset and $w = 1{,}000$ days for Synthetic dataset. With such window sizes, it is guaranteed that we can discover all relevant patterns.

5.4.1 Effectiveness and Pattern Meaning

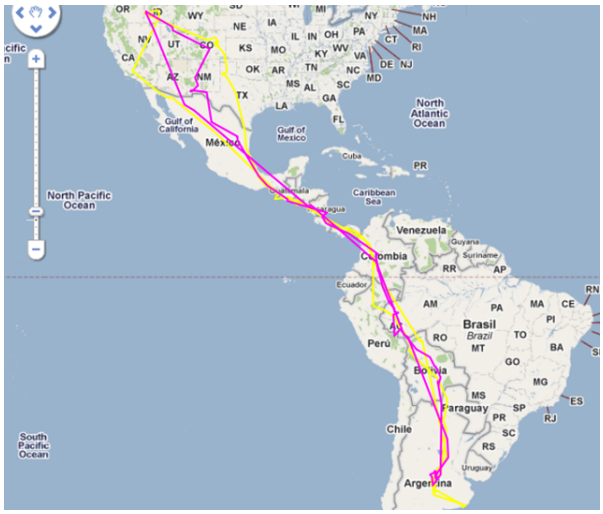
The effectiveness of interesting maximal rGpatterns can be demonstrated through our online demo system. Figures 5.9, 5.10 show the number of interesting maximal rGpatterns extracted from the datasets. One of the extracted patterns from Swainsoni dataset is illustrated in Figure 5.6c. Each color represents a Swainsoni trajectory segment involved in the pattern. Additionally, each place mark is a cluster center with the number of objects information. For clarity sake, we only report the locations where the number of objects changes.

To illustrate the knowledge expressed by maximal rGpatterns, we also show some of extracted closed swarms and convoys from our system [4] [8] in Figures 5.6a, b. We can consider that the closed swarm and the convoy show two or three objects moving together by the time. Even though they are interesting since they express the relationship between several objects, they fail to describe the moving behavior of the class of Swainsonies.

Distinguish from previous work, by proposing interesting maximal rGpattern, 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 5.6c, 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* $\{\mathtt{Oregon}\rangle\ \mathtt{Nevada}\rangle\ \mathtt{Utah}\rangle\ \mathtt{Arizona}\rangle\ \mathtt{Mexico}\rangle\ \mathtt{Colombia}\}$". 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, Panama is also important since all objects are together before arriving Colombia, South America. That is very useful knowledge for planning research and conservation activities such as migration analysis, telemetry attaching, counting, etc.

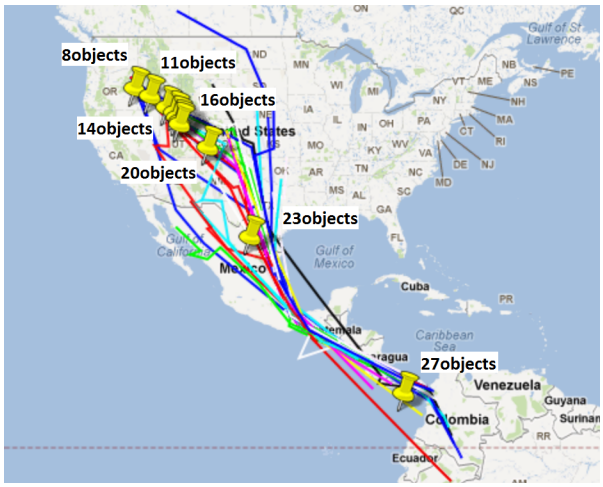
The discovery of the interesting maximal rGpatterns on animal migration datasets provides useful information for biologists to better understand and examine the relationship and habits of these moving objects. None of the other moving object patterns is able to ex-

4. http://www.lirmm.fr/~phan/index.jsp



(a) One of extracted closed swarms.

(b) One of extracted Convoy.



(c) One of extracted interesting maximal rGpatterns $C^{\geq}$.

Figure 5.6: An example of extracted patterns from Swainsoni dataset (best viewed in color).

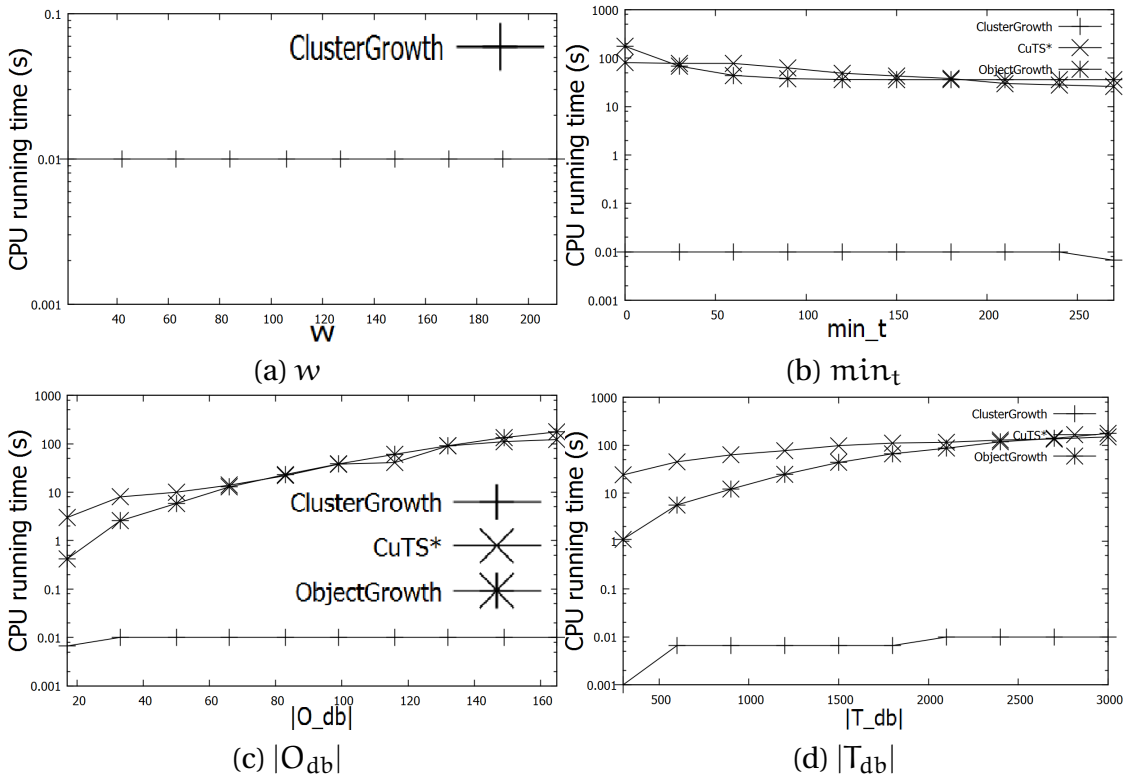


Figure 5.7: Running time on Buffalo Dataset.

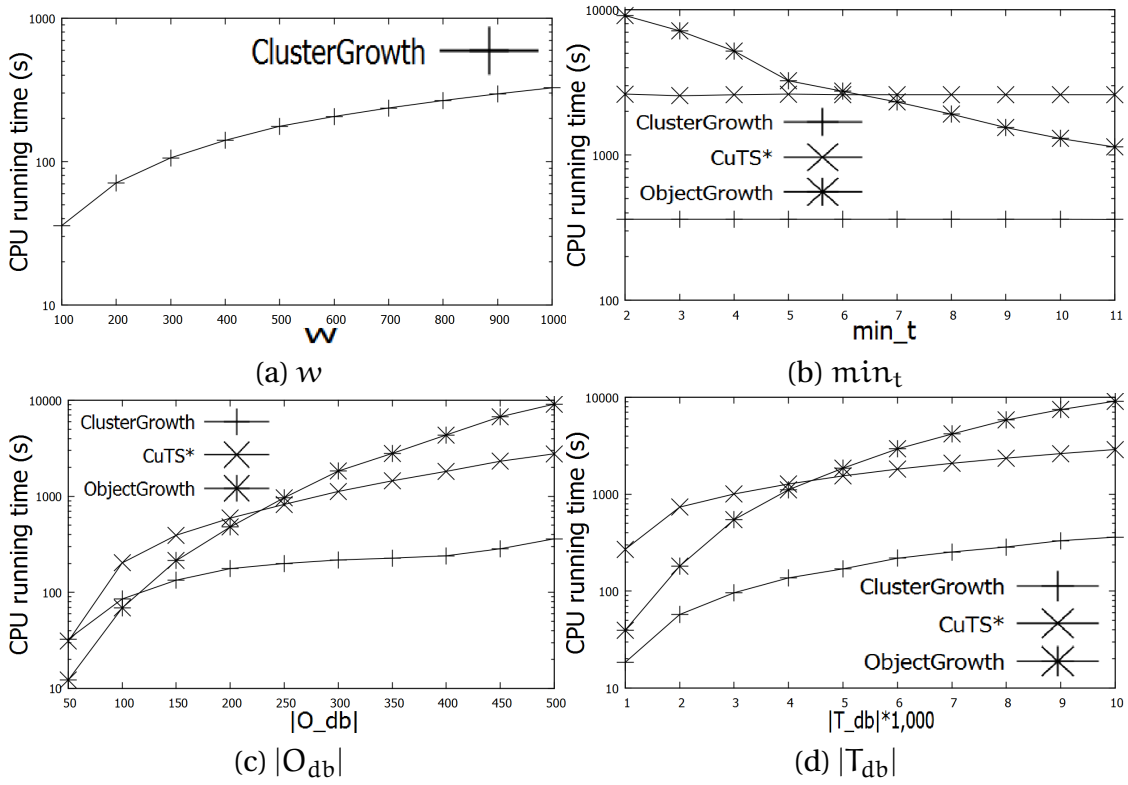
tract this knowledge. They focus on an unchanged group of moving objects during a time period while interesting maximal rGpattern relaxes this constraint and captures moving trend patterns from a graduality point of view.

5.4.2 Parameter Sensitiveness

To show the parameter sensitiveness and efficiency of the proposed algorithm, as in [6], 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 with slow moving speed (250). 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 Figures 5.7a, 5.8a, 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 (see Figures 5.9a, 5.10a). 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 as well as patterns. Note that, the

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



(a) w (b) min_t

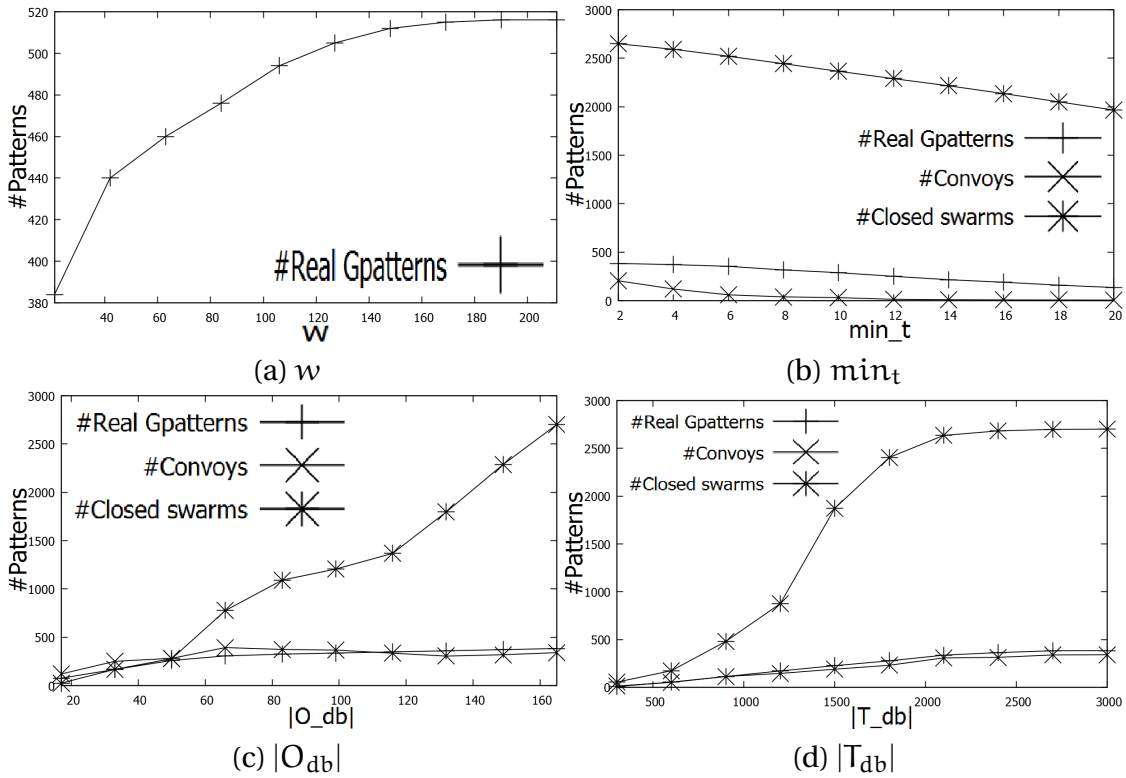
(c) $|O_{db}|$ (d) $|T_{db}|$

Figure 5.8: Running time on Synthetic Dataset.

ClusterGrowth algorithm is very efficient while the Buffalo dataset is not large enough to see the difference in terms of ClusterGrowth execution time.

Sensitiveness w.r.t min_t. Figures 5.7b, 5.8b show 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 (Figures 5.9b, 5.10b). Meanwhile, ClusterGrowth and CuTS* only use min_t at the pattern reporting step without any pruning rule for min_t. Therefore, be 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}. Figures 5.7c-d, 5.8c-d show the sensitiveness in the sizes of O_{db} and T_{db}. Once again, ObjectGrowth is the most sensitive algorithm. 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 (see Figures 5.9c-d, 5.10c-d). 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



(a) w (b) min_t

(c) $|O_{db}|$ (d) $|T_{db}|$

Figure 5.9: Number of patterns on Buffalo Dataset. For conciseness sake, #Interesting Maximal rGpatterns is denoted as #rGpatterns.

$|T_{db}|$ 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 $|T_{db}|$.

To summarize, ClusterGrowth is effective to extract interesting maximal rGpatterns which are novel and useful movement patterns. Additionally, ClusterGrowth is less sensitive in terms of parameter setting and database sizes than the other algorithms (i.e. ObjectGrowth) in almost cases. Furthermore, with the purpose to extract the complete set of interesting maximal rGpatterns, ClusterGrowth is competitive in time efficiency to state of the art algorithms (see Figures 5.7, 5.8).

5.5 Mining Representative Gradual Trajectory Patterns

Until now, with the CLUSTERGROWTH algorithm we can efficiently extract all interesting maximal rGpatterns efficiently. However, as usual 1) the number of rGpatterns can be large and 2) the result contains very redundant information. Thus it is difficult to use

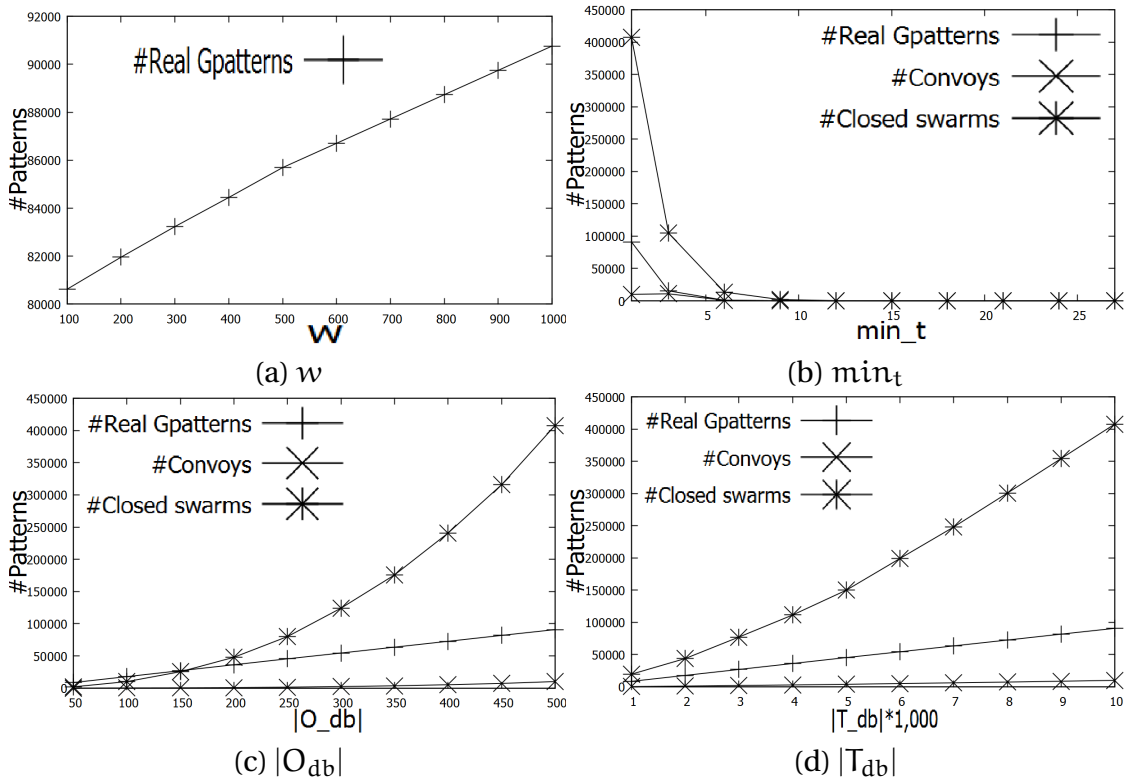


Figure 5.10: Number of patterns on Synthetic Dataset. For conciseness sake, #Interesting Maximal rGpatterns is denoted as #rGpatterns.

and analyze the extracted patterns. Therefore, eliminating the number of rGpatterns and extracting representative ones is an emerging task in many real world cases.

Mining informative patterns can be classified into 3 main lines: MDL-based, 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. [3] for data clustering problem. This idea was also explored by Keogh et al. [20], who propose to use compressibility as a measure of distance between two sequences.

In the second research line, the significance of patterns is assessed by using a standard statistical hypothesis assuming that the data follows the null hypothesis. If a pattern passes the test it is considered significant and interesting. For instance, A. Gionis et al. [5] 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. [25].

Another direction looks for interesting sets of patterns that compress most the given data (i.e. MDL principle). Examples of this direction include the Krimp algorithm [31] for itemset data, the algorithms for sequence data [21] and the algorithms for moving object data [11].

Like many pattern set mining approaches [34] [35] [36], Krimp follows a straightforward two-phase approach to mining code tables. First, it mines a collection of frequent itemsets. Second, it considers these candidates in a static order, accepting a pattern if it improves compression. This simplicity has some drawbacks. First of all, mining candidates is expensive but as most candidates will be rejected and therefore this step is quite wasteful. Second, by considering candidates only once, and in a fixed order, Krimp sometimes rejects candidates that it could have used later on.

To break through these issues, K. Smets et al. [27] propose SLIM which is an improvement of KRIMP. SLIM starts with a set of single items in code table CT. Every iteration, it considers all pairwise combinations of $X, Y \in CT$ as candidates in descending order of gain. Iteratively, if a candidate improves the compression then it will be added into the code table CT. Next, elements in CT will be reconsidered and the candidate list is updated. The process continues until no candidate decreases the total compressed size.

In sequence data context, H. T. Lam et al. propose GOKRIMP [21] which greedily constructs a pattern by extending a candidate with the most frequent correlated item such that compression is improved. In this work, we will focus on designing an MDL-based approach to extract top-K representative rGpatterns.

5.5.1 Problem Statement

First of all, let us start explaining the MDL principle in the following:

Definition 20. *Hypothesis. A hypothesis $\mathcal{D}$ is a set of patterns $\mathcal{D} = \{p_1, p_2, \ldots, p_h\}$.*

Given a scheme S, let $L_S(D)$ be the description length of hypothesis $\mathcal{D}$ and $L_S(O_{db}|D)$ 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 $\mathcal{D}$ and the encoding scheme S such that $L_S(O_{db}) = L_S(\mathcal{D}) + L_S(O_{db}|\mathcal{D})$ 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 $\mathcal{D}$ is denoted as $L_{\mathcal{D}}(O_{db}) = L(\mathcal{D}) + L(O_{db}|\mathcal{D})$.

In this chapter, the hypothesis is considered as a dictionary of rGpatterns $\mathcal{D}$. Furthermore, as in [21] [11], we assume that any number or character in data has a fixed bit length representation which requires a unit memory cell. In our context, the description length of a dictionary $\mathcal{D}$ can be calculated as the total lengths of the patterns and the number of patterns, i.e. $L(\mathcal{D}) = \sum_{C^* \in \mathcal{D}} |C^*| + |\mathcal{D}|$. Furthermore the length of the data O_{db} when encoded with the help of dictionary $\mathcal{D}$ can be calculated as $L(O_{db}|\mathcal{D}) = \sum_{o \in O_{db}} |o|$.

The problem of finding compression rGpatterns can be formulated as follows:

Definition 21. *Compression rGpattern Problem. Given a moving object data O_{db}, a set of rGpatterns $F = \{C_1^*, \ldots, C_m^*\}$. Discover an optimal dictionary $\mathcal{D}^\# \subseteq F$ s.t.*

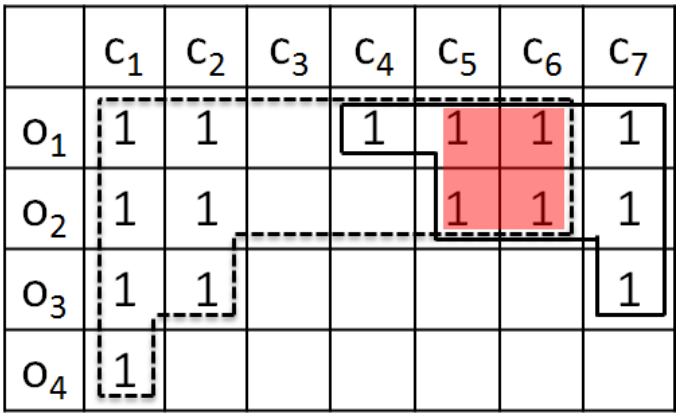


Figure 5.11: An example of two types of rGpatterns. c and o respectively are cluster and object, $(o_i, c_j) = 1$ means o_i belongs to cluster c_j. Blurred rectangle is the overlapping part between two rGpatterns.

$$\mathcal{D}^{\#} = \arg\min_{\mathcal{D}} (L^{\#}_{\mathcal{D}}(O_{db})) = \arg\min_{\mathcal{D}} (L^{\#}(\mathcal{D}) + L^{\#}(O_{db}|\mathcal{D}))$$

A key issue in designing an MDL-based algorithm is: how can we encode data given a dictionary? The issue here is that we have two different types of patterns (i.e. $C^{\geq}$ and $C^{\leq}$) and traditional algorithms [31] [27] [21] do not supply multi-pattern types in the dictionary that may lead to losing interesting ones. Furthermore, enforcing non-overlapping patterns may result in not optimal compression. To deal with these challenges, we propose an encoding scheme which is able to deal with different types of rGpatterns and the overlapping among them.

5.5.2 Encoding Scheme

Before discussing our encoding for moving object data given rGpatterns, we revisit the encoding scheme used in the Krimp algorithm [31]. 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.

In this chapter we use a similar dictionary-based encoding scheme. Given a dictionary consisting of rGpatterns $\mathcal{D} = \{C^*_1, \ldots, C^*_m\}$, an object $o \in O_{db}$ containing a list of clusters is encoded by replacing instances of any pattern C^*_i in o with pointers to the dictionary. An important difference in our context is that we have two types of rGpatterns which have their own characteristics. Indeed, an object o can involve in the some first part of a $C^{\leq}$ or in the some last part of a $C^{\geq}$.

For instance, see Figure 5.11a and Table 5.3, we can consider that o_2 joins the $C^{\geq}_2 = c_4c_5c_6c_7$ at $c_5c_6c_7$. While, the $C^{\leq}_1 = c_1c_2c_3$ occurs in o_3 at c_1c_2.

Table 5.3: An illustrative example of data and dictionary in Figure 5.11. $\bar{1}$ and $\bar{2}$ respectively are pattern types: $C^{\geq}$ and $C^{\leq}$.

O_{db}	Encoded O_{db}	Dictionary $\mathcal{D}$
$o_1 = c_1c_4c_5c_6c_7$	$o_1 = [C_1,0][C_2,0]$	
$o_2 = c_1c_2c_3c_5c_6c_7$	$o_2 = [C_1,2][C_2,1]$	$C_1^{\leq} = c_1c_2c_3, \bar{2}$
$o_3 = c_1c_2c_7$	$o_3 = [C_1,1][C_2,3]$	$C_2^{\geq} = c_4c_5c_6c_7, \bar{1}$
$o_4 = c_1$	$o_4 = [C_1,0]$	

Property 14. ***(Encoding Property)****. Given an object* o *which contains a list of clusters and a rGpattern* $C^* = c_1 \ldots c_n$. C^* *occurs in* o *or* o *contributes to* C^* *if:*

$$\begin{cases} (1) : * = \geq, \exists i \in [1,n] | \forall j \geq i, c_j \in o. \\ (2) : * = \leq, \exists i \in [1,n] | \forall j \leq i, c_j \in o. \end{cases} \tag{5.6}$$

Proof. Case (1): we have $c_i \subseteq c_{i+1} \subseteq \ldots \subseteq c_n$. Additionally, $o \in c_i$. Consequently, $o \in c_{i+1}, \ldots, c_n$ and therefore $\forall j \geq i, o \in c_j$. In Case (2): we have $c_1 \supseteq c_2 \supseteq \ldots \supseteq c_{i-1}$. Additionally, $o \in c_{i-1}$. Consequently, $o \in c_1, \ldots, c_{i-1}$ and therefore $\forall j \leq i, o \in c_j$. □

For instance, see Table 5.3, we can see that for each pattern, we need to store an extra bit to indicate the pattern type. Additionally, by applying Property 14, in the object o we also need to store an additional index with the pointer 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 $C^{\geq}$ (resp. $C^{\leq}$).

As an example, consider dictionary $\mathcal{D}$ in Table 5.3. Using $\mathcal{D}$, o_2 can be encoded as $o_2 = [C_1,2][C_2,1]$ where 2 in $[C_1,2]$ indicates the cluster at index 2 in $C_1^{\leq}$, i.e. c_3, and 1 in $[C_2,1]$ indicates the cluster at index 2 in $C_2^{\geq}$, i.e. c_5.

Until now, we have already presented the encoding scheme for an object o given a rGpattern C^*. Furthermore, the encoding scheme also allows overlapping rGpatterns in the data compression. For instance, in Figure 5.11, $C_2^{\geq} = c_4c_5c_6c_7$ and $C_1^{\leq} = c_1c_2c_5c_6$ overlap each other. Let us assume that $C_2^{\geq} \in \mathcal{D}$ and $C_1^{\leq} \in F$, so object o_1 will be first encoded given $C_2^{\geq}$ s.t. $o_1 = c_1[C_2,0]$. We can consider that even $C_1^{\leq}$ overlaps $C_2^{\geq}$ at c_5c_6, $C_1^{\leq}$ can still be used to encode o_1 s.t. $o_1 = [C_1,0][C_2,0]$.

Data description length computation. Until now, we have defined an encoding scheme for rGpatterns. The description length of the dictionary in Table 5.3 is calculated as $L(\mathcal{D}) = |C_1^{\leq}| + 1 + |C_2^{\geq}| + 1 + |\mathcal{D}| = 3 + 1 + 4 + 1 + 2 = 11$. Similarly, description length of o_2 is $L(o_2|\mathcal{D}) = |[C_1^{\leq}, 2]| + |[C_2^{\leq}, 1]| = 4$.

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

5.5.3 Complexity Analysis

This section discusses the complexity of the mining compression rGpattern problem. The main result states that finding optimal dictionary $\mathcal{D}$ is NP-Hard.

Lemma 6. *Finding compression rGpattern problem and the optimal dictionary $\mathcal{D}$ is NP-Hard.*

Proof. Given a set of elements $\{1,2,\ldots,m\}$ (called the universe) and n sets whose union comprises the universe, the set cover problem [26] is to identify the smallest number of sets whose union still contains all elements in the universe.

We can easily reduce the mining optimal compression rGpattern problem to the set cover problem by setting each '1' in the dataset as an element, the universe as a set of all elements in the dataset, each rGpattern as a set of elements, i.e. '1's, and the encoding function as the number of '1's in a rGpattern. The problem now becomes finding the smallest number of rGpatterns whose union still contains all the elements, i.e. '1's, in the universe. In doing so, finding the optimal compression rGpatterns is as hard as solving the corresponding instance of the set cover problem. This proves the NP-hardness property of the given problem. □

5.5.4 Mining top-K Representative rGpatterns

Since mining optimal dictionary $\mathcal{D}$ is NP-Hard, we propose two heuristic algorithms, named CompoGp and DiCompoGp. The first one is inspired by the idea of the Krimp algorithm and it is described in the following.

The Two-Phase Approach

The greedy approach takes as input a database O_{db}, a set of pattern candidates F and a parameter K. The result is the optimal dictionary which encodes O_{db} best. Now, at each iteration of CompoGp, we select candidate C^* which compresses the database best. Next, C^* will be added into the dictionary $\mathcal{D}$ and then the database O_{db} will be encoded given C^*. The process is repeated until we obtain K rGpatterns in $\mathcal{D}$.

For each candidate $C^* \in F$ at each iteration, $\overline{O}_{C^*}$ and O_{C^*} respectively are the set of objects that do not contribute to C^* and the set of objects contribute to C^*. The number of clusters in $o \in O_{C^*}$ can be replaced by a pointer to C^* is denoted as $involve_{o,C^*}$. For instance, see Table 5.3, $involve_{o_1,C_2^{\geq}} = 4$ since c_4, c_5, c_6 and c_7 are replaced by $[C_2, 0]$.

The compression gain which is the number of memory cells we earned when adding C^* into dictionary $\mathcal{D}$ can be defined as $gain(C^*,\mathcal{D}) = L_{\mathcal{D}}(O_{db}) - L_{\mathcal{D}\cup C^*}(O_{db})$. $gain(C^*,\mathcal{D})$ can be computed as follows:

Property 15. ***(Compression Gain).*** *Given a dictionary $\mathcal{D}$, a rGpattern $C^* \in F$. $gain(C^*,\mathcal{D})$ is computed as:* $gain(C^*,\mathcal{D}) = \sum_{o\in O_{C^*}} involve_{o,C^*} - (2 \times |O_{C^*}| + |C^*| + 2)$

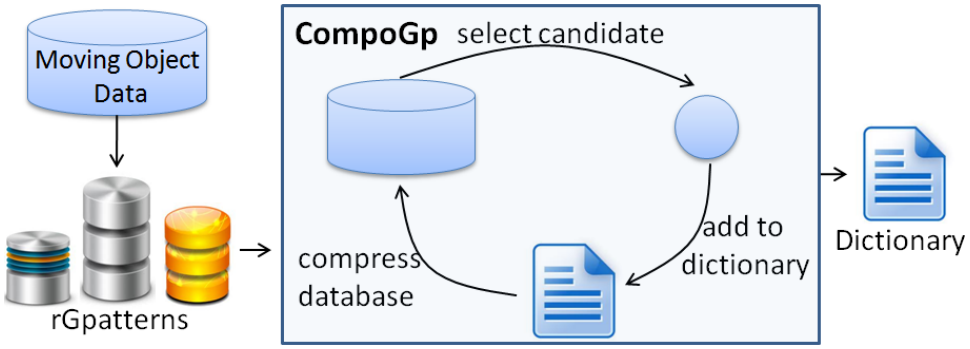


Figure 5.12: COMPOGP algorithm in action.

Proof. After construction we have $L_{\mathcal{D}\cup C^*}(O_{db}) = L(\mathcal{D} \cup C^*) + L(O_{db}|\mathcal{D} \cup C^*) = (L(\mathcal{D}) + |C^*| + 2) + L(\overline{O}_{C^*}|\mathcal{D}) + L(O_{C^*}|\mathcal{D} \cup C^*)$. Note that $L(\overline{O}_{C^*}|\mathcal{D}) = L(\overline{O}_{C^*}|\mathcal{D} \cup C^*)$. Furthermore, $\forall o \in O_{C^*} : L(o|\mathcal{D} \cup C^*) = L(o|\mathcal{D}) - involve_{o,C^*} + 2$. Thus $L(O_{C^*}|\mathcal{D} \cup C^*) = L(O_{C^*}|\mathcal{D}) - \sum_{o\in O_{C^*}} involve_{o,C^*} + 2 \times |O_{C^*}|$. Therefore we have $L_{\mathcal{D}\cup C^*}(O_{db}) = L(\mathcal{D}) + L(O_{db}|\mathcal{D}) - \sum_{o\in O_{C^*}} involve_{o,C^*} + 2 \times |O_{C^*}| + |C^*| + 2$. Consequently, we have $gain(C^*, \mathcal{D}) = L_{\mathcal{D}}(O_{db}) - L_{\mathcal{D}\cup C^*}(O_{db}) = \sum_{o\in O_{C^*}} involve_{o,C^*} - (2 \times |O_{C^*}| + |C^*| + 2)$. □

By applying Property 15, we can compute the compression gain when adding a new rGpattern C^* into the dictionary $\mathcal{D}$. Essentially, the $gain(C^*, \mathcal{D})$ depends on how many items an object $o \in O_{C^*}$ contribute to C^*, the number of objects in O_{C^*}, and the size of C^* and all of them can be computed by scanning C^* with objects $o \in O_{C^*}$. COMPOGP is presented in *Algorithm 7*.

Directly Mining Representative rGpatterns

Although the COMPOGP algorithm enables us to mine representative rGpatterns, it has some drawbacks: 1) Mining candidates is expensive. As more candidates correspond to a larger search space, lower min_t correspond to better final results. However, it is difficult to keep the number of candidates feasible for large and dense databases and especially a small drop of min_t can lead to an enormous increase in rGpatterns. Moreover, as most candidates will be rejected, this step is quite wasteful. 2) We need to scan the whole search space F at every iteration that is a very expensive task and time consuming since most of them are redundant. Thus, it quickly becomes infeasible for larger candidate collections.

With DICOMPOGP, we address these issues and propose an efficient algorithm for mining representative rGpatterns *directly* from data. In order to do so, we first present a way to efficiently compute the compression gain for candidates C^*.

Directly Evaluating Candidate Compression Gain. given an object $o_i \in O_{C^*}$ and a rGpattern C^*, the basic idea is to compute the $involve_{o_i,C^*}$, i.e. in Property 15, directly

Algorithm 7: CompoGp Algorithm

Input : O_{db}, C_{db}, set of patterns F, int K, int min_t, int min_w, $*$
Output: Compressing patterns $\mathcal{D}$

begin
  $F \longleftarrow ClusterGrowth(C_{db}, min_t, min_w, *)$;
  $\mathcal{D} \longleftarrow \emptyset$;
  while $|\mathcal{D}| < K$ **do**
    foreach $p \in F$ **do**
      $L^*(O_{db}|p) \longleftarrow Benefit(O_{db}, p)$;
    $p^* \longleftarrow \arg\max_p L^*(O_{db}|p)$;
    $\mathcal{D} \longleftarrow p^*$; $F \longleftarrow F \setminus \{p^*\}$;
    Replace all instances of p^* in O_{db} by its pointers;
  output $\mathcal{D}$;
Benefit(O_{db}, p)
begin
  $b \longleftarrow 0$;
  foreach $o \in O_{db}$ **do**
    if $p.involved(o) = true$ **then**
      $b \longleftarrow b + benefit(o,p) - 2$;
  $b \longleftarrow b - (|p| + 2)$;
  output b;

without scanning o_i on C^*. Let us assume that there is a list of clusters $C^{used}_{o_i} \subseteq C^*$ which have been replaced in the compression of o_i before, $last(C^{used}_{o_i}) = c_j$ is the last indexed cluster in $C^{used}_{o_i}$ and $index(c_j, C^{\geq}) = j$ (see Figure 5.13). By applying the encoding scheme for $C^{\geq}$, i.e. Property 14, only $c_{j+1}, \ldots, c_n$ can be used to compress o_i and therefore $involve_{o_i, C^{\geq}} = |C^{\geq}| - index(c_j, C^{\geq}) = n - j$. The $gain(C^{\geq}, \mathcal{D})$ can be computed as follows:

Property 16. ***(Compression Gain for*** $C^{\geq}$***).*** *Given a dictionary* $\mathcal{D}$*, a rGpattern* $C^{\geq} \in F$*.* $gain(C^{\geq}, \mathcal{D})$ *is computed as:* $\sum_{o \in O_{C^{\geq}}} (|C^{\geq}| - index(last(C^{used}_o), C^{\geq})) - (2 \times |O_{C^{\geq}}| + |C^{\geq}| + 2)$

In the implementation, instead of storing a list of clusters $C^{used}_{o_i}$, we store a list of objects $O^{used}_{c_j}$ each of them already replaced c_j by some pointers before (Figure 5.13). The gain computation function for $C^{\geq}$ is as follows:

1. For each cluster $c \in C^{\geq}$
   a. For each object $o \in c$, $involve_{o, C^{\geq}} \mathrel{+}= 1$.
   b. For each object $o \in O^{used}_c$, $involve_{o, C^{\geq}} = 0$.
2. $gain(C^{\geq}, \mathcal{P})$ is computed as $\sum_{o \in O_{C^{\geq}}} involve_{o, C^{\geq}} - (2 \times |O_{C^{\geq}}| + |C^{\geq}| + 2)$.

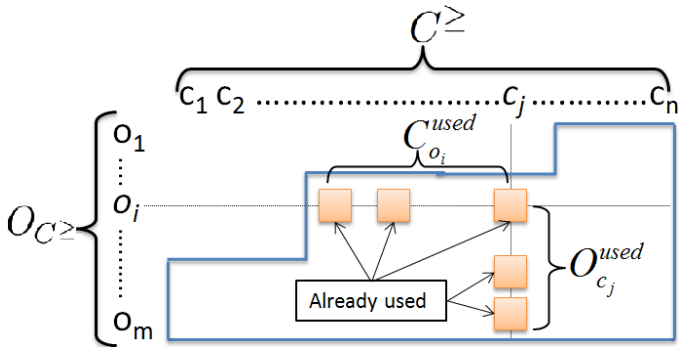


Figure 5.13: An example of directly evaluating compression gain of $C^{\geq}$. $C^{used}_{o_i}$ and $O^{used}_{c_j}$ are lists of blurred rectangles.

In this function, $involve_{o,C^{\geq}}$ will be increased by 1 whenever o belongs to the cluster c. Then for each o which already replaced c by some pointer, $involve_{o,C^{\geq}}$ will be reset to 0 since all of the clusters, before c in the object o, cannot be used in the compression. Finally, the $gain(C^{\geq},\mathcal{D})$ is easily computed based on $involve_{o,C^{\geq}}$.

Similarly, we also have a property for $C^{\leq}$ and the difference is that $involve_{o,C^{\leq}} = index(first(C^{used}_o), C^{\leq})$.

Property 17. ***(Compression Gain for*** $C^{\leq}$***).*** *Given a dictionary* $\mathcal{D}$*, a rGpattern* $C^{\leq} \in F$. $gain(C^{\leq},\mathcal{D})$ *is computed as:* $gain(C^{\leq},\mathcal{D}) = \sum_{o \in O_{C^{\geq}}} index(first(C^{used}_o), C^{\leq}) - (2 \times |O_{C^{\geq}}| + |C^{\geq}| + 2)$.

The DICOMPOGP **Algorithm.** Until now, given a rGpattern, we can evaluate the compression gain without scanning the database. In this section, we will present an heuristic solution to directly extract top-K representative rGpatterns.

First of all, let us denote a list of clusters as $\mathcal{C} = \{c_1, \ldots, c_n\}$ such that $c_1 \supseteq \ldots \supseteq c_n$. We have that

- $\forall i \in [1, n-1]$:
- if $1 \leq t(c_{i+1}) - t(c_i) \leq w$ then $\mathcal{C}$ is a $C^{\leq}$.
- if $1 \leq t(c_i) - t(c_{i+1}) \leq w$ then $\overleftarrow{\mathcal{C}} = \{c_n, \ldots, c_1\}$ is a $C^{\geq}$.

From now, $\mathcal{C} = \{c_1, \ldots, c_n\}$ will be consider as a candidate which is refinable to rGpatterns and furthermore c_1 is the largest cluster in $\mathcal{C}$.

Basic idea of DICOMPOGP**.** Given a largest cluster $c_i \in C_{db}$ which has not been used for data covering before. We first extract all the candidates $\mathcal{C} = \{c_i, c_{i+1}, \ldots, c_{i+n}\}$ and compute their compression gains by applying Properties 16 and 17. The best candidate $\mathcal{C}$ which has the highest compression gain will be chosen to cover the data. If compression is improved, the candidate is accepted otherwise it is rejected. If accepted, all the clusters $c \in \mathcal{C}$ will be

Table 5.4: An illustrative example of moving object data.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
o_1				1	1			
o_2		1	1	1	1			1
o_3			1	1	1		1	
o_4	1	1	1		1	1	1	

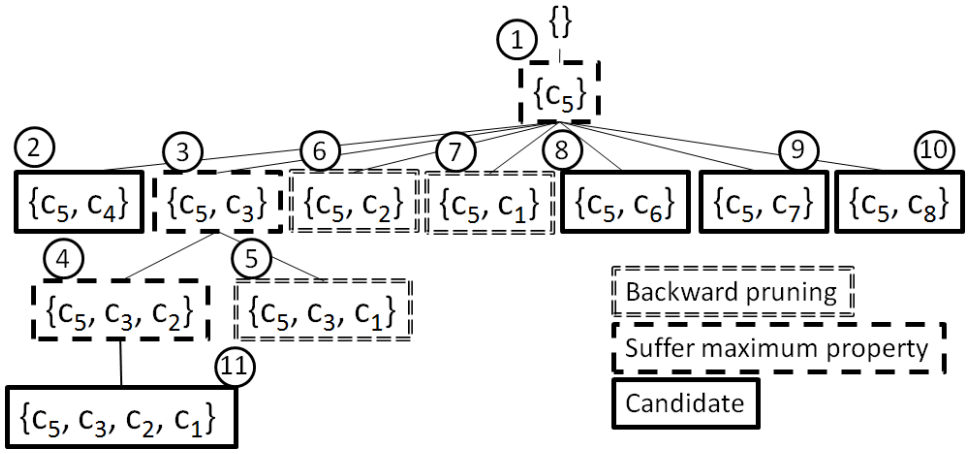


Figure 5.14: An illustration of DICOMPOGP in Table 5.4. c_5 is the largest cluster among $c_1, \ldots, c_8$.

considered as used clusters. The process continues with the other largest clusters and will end when there are K interesting maximal rGpatterns in $\mathcal{D}$.

DICOMPOGP also employs depth-first-search approach on the cluster set C_{db}. To reduce the search space, we adapt the proposed pruning rules in the CLUSTERGROWTH. Let us first consider the *Di-Graduality Pruning* rule as follows:

Property 18. ***(Di-Graduality Pruning)***. *Given a candidate* $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ *with* $c_1 \supseteq \ldots \supseteq c_n$, *a cluster* c', *and a window size* w. *There is no strict superset* $\mathcal{C}' \supseteq (\mathcal{C} \cup c')$ *s.t.* $\mathcal{C}'$ *(resp.* $\overleftarrow{\mathcal{C}'}$*) is a rGpattern if:*

$$\begin{cases} \overleftarrow{\mathcal{C}} = C^{\geq} : (t(c_n) - t(c') > w) \vee (c_n \not\supseteq c') \\ \mathcal{C} = C^{\leq} : (t(c') - t(c_n) > w) \vee (c_n \not\supseteq c') \end{cases} \tag{5.7}$$

For instance, see Table 5.4 and Figure 5.14, the node $\{c_5, c_4, c_3\}$ is pruned since it suffers the graduality, i.e. $c_4 \not\supseteq c_3$. For conciseness reason, we do not show the nodes pruned by Di-Graduality Pruning. Furthermore, the *Di-Backward Pruning* can be defined as follows:

Property 19. ***(Di-Backward Pruning)***. *Given a candidate* $\mathcal{C}=\{c_1,c_2,\ldots,c_n\}$ *with* $c_1 \supseteq \ldots \supseteq c_n$. *If there exists a cluster* c' *such that:*

$$\begin{cases} \overleftarrow{\mathcal{C}} = C^{\geq} : t(c_1) > t(c') > t(c_n) \wedge \mathcal{C}' = \overleftarrow{\mathcal{C} \cup c'} \\ \mathcal{C} = C^{\leq} : t(c_1) < t(c) < t(c_n) \wedge \mathcal{C}' = \mathcal{C} \cup c' \end{cases} \tag{5.8}$$

and $\mathcal{C}'$ *satisfies the condition 2 - Definition 17 (i.e. graduality condition) then any supersets of* $\mathcal{C}$ *are not maximal rGpatterns. Thus,* $\mathcal{C}$ *subtrees need to be pruned.*

For instance, see Figure 5.14, all the subtrees of the node labeled $5\{c_5,c_3,c_1\}$ are pruned since $\overleftarrow{\{c_5,c_3,c_2,c_1\}}=\{c_1,c_2,c_3,c_5\}$ satisfies the graduality condition. Similarly, we also have the *Di-Actual Maximum rule* as follows:

Property 20. ***(Di-Actual Maximum Rule)***. *Given a candidate* $\mathcal{C}=\{c_1,\ldots,c_n\}$ *with* $c_1 \supseteq \ldots \supseteq c_n$. *If there exists a cluster* c' *such that:*

$$\begin{cases} \overleftarrow{\mathcal{C}} = C^{\geq} : t(c_n) > t(c') \wedge \mathcal{C}' = \overleftarrow{\mathcal{C} \cup c'} \\ \mathcal{C} = C^{\leq} : t(c_n) < t(c') \wedge \mathcal{C}' = \mathcal{C} \cup c' \end{cases} \tag{5.9}$$

and $\mathcal{C}'$ *satisfies the condition 2 - Definition 17 (i.e. graduality condition) then* $\mathcal{C}$ *is not maximal.*

For instance, the node labeled $3\{c_5,c_3\}$ is not maximal since $\overleftarrow{\{c_5,c_3,c_2\}}=\{c_2,c_3,c_5\}$ also satisfies the graduality condition. Similar to Theorem 1, we also have that if a candidate passes all the pruning rules then it is an interesting maximal rGpattern. The adaptation of the three pruning rules above efficiently eliminates a large number of useless candidates.

For instance, see Table 5.4 and Figure 5.14, we have 5 candidates $\big\{\{c_5,c_3,c_2,c_1\}, \{c_5,c_4\},\{c_5,c_6\},\{c_5,c_7\},\{c_5,c_8\}\big\}$. Since $\mathcal{C}=\{c_5,c_3,c_2,c_1\}$ has the highest compression gain and the compression is improved, it is considered as the best candidate. $\overleftarrow{\mathcal{C}}$ is a representative interesting maximal rGpattern $C^{\geq}$. We give the pseudo-code in Algorithm 8.

Memory consumption analysis. Let us denote $F=\sum_{p\in F}|p|$, $O_{db}=\sum_{o\in O_{db}}|o|$, $C_{db}=\sum_{c\in C_{db}}|oK(c)|$, $\mathcal{D}=\sum_{C^*\in\mathcal{P}}|C^*|$. Additionally, $|F|,|T_{db}|$ and $|C_{db}|$ respectively are the number of candidates in F, the number of timestamps in T_{db} and the number of clusters in C_{db}.

For the COMPOGP, we need to store $F, O_{db}, C_{db}, T_{db}$ and $\mathcal{D}$ in memory. Thus, the needed memory is $F+O_{db}+C_{db}+|T_{db}|+\mathcal{D}$. In the other hand, instead of storing all the candidates F, at a specific moment DICOMPOGP only needs to store a set of candidates $\mathcal{C}$ containing the largest cluster c_i. In average, the number of candidates which contain a specific cluster c_i is $\frac{F}{|C_{db}|}$ and average size of a candidate is $\frac{F}{|F|}$. Thus, the average size of all the candidates $\mathcal{C}$ is $\frac{F\times F}{|C_{db}|\times|F|} \ll F$ since $|C_{db}|\times|F| \gg F$. Additionally, $F \gg O_{db}, C_{db}, \mathcal{D}, |T_{db}|$ usually. Consequently, the average needed memory for DICOMPOGP is $\frac{F\times F}{|C_{db}|\times|F|}+O_{db}+C_{db}+|T_{db}|+\mathcal{D}$ which is greatly smaller than the one of COMPOGP.

Algorithm 8: DiCompoGp Algorithm

Input : Database O_{db}, cluster database C_{db}, int K, int min_t, int w
Output: Compressing patterns $\mathcal{P}$

```
1  begin
2      P ⟵ ∅;
3      while |P| < K do
4          c1 ⟵ biggestCluster(C_db);
5          if c1 ≠ ∅ then
6              Cands ⟵ extractCandidate(C_db, c1, min_t, w);
7              C ⟵ argmax_{C'∈Cands}(compression_gain(C'));
8          else
9              output P;
10         P ⟵ P ∪ C;
11         Replace all instances of C in O_db by its pointers;
12         Remark all clusters in C as used clusters in C_db;
13     output P;
```

5.6 Experimental Results on Mining Representative rGpatterns

In the comparison, we compare the set of patterns produced by DICOMPOGP, COMPOGP and CLUSTERGROWTH [6]. Additionally, to retrieve all the patterns, in the reported experiments the default value of min_t is 1.

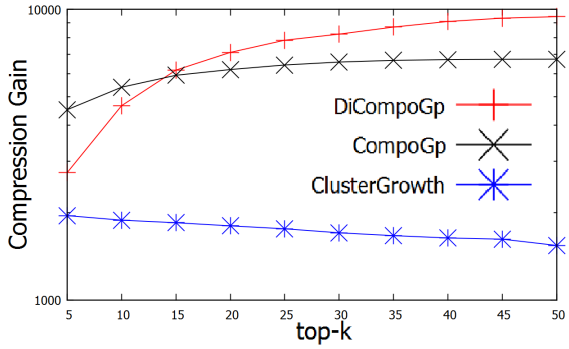
To show the parameter sensitiveness and efficiency of the proposed algorithms, as in [23] [8] [12], we also generate a large synthetic dataset using Brinkhoff's network [7]-based generator of moving objects. We generate 500 objects ($|O_{db}| = 500$) for 10^4 timestamps ($|T_{db}| = 10^4$) using the generator's default map, i.e. Copenhagen road network, with slow moving speed. There are 5×10^6 points in total. DBScan ($minpts = 3, eps = 300$) is applied to obtain clusters at each timestamp. Note that $w = 1{,}000$ days for this synthetic dataset.

Compressibility

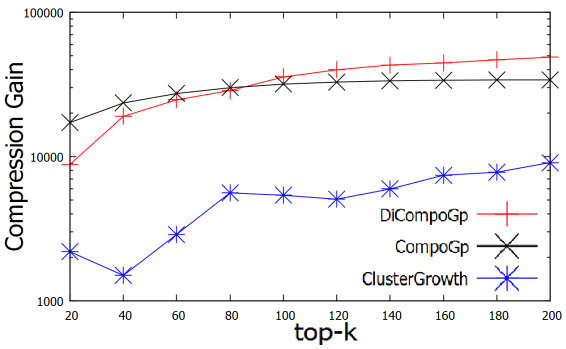
We measure the compressibility of the algorithms by using their top-K patterns as dictionaries for encoding the data. The compression gain could be calculated as the sum of the compression gain returned after each step. Additionally, to illustrate the difference be-

6. Note that in [10], the CLUSTERGROWTH algorithm outperforms traditional algorithms such as CuTS* [18] (convoy mining) and ObjectGrowth [23] (closed swarm mining) in terms of performance and thus we will only consider the CLUSTERGROWTH and not all.

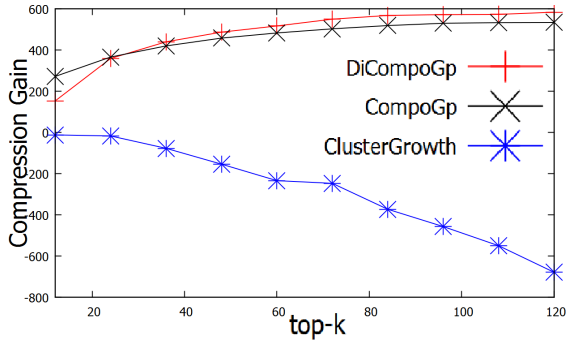
7. http://iapg.jade-hs.de/personen/brinkhoff/generator/



(a) Swainsoni dataset.



(b) Buffalo dataset.



(c) Synthetic dataset.

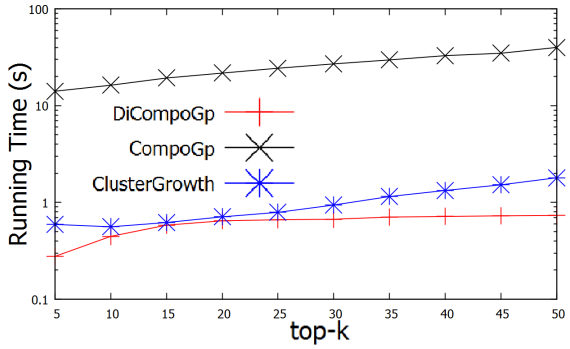
Figure 5.15: Compressibility (higher is better) of different algorithms.

tween MDL-based approaches and standard support-based approaches, we also employ the set of top-K highest covering area rGpatterns which stands for the CLUSTERGROWTH algorithm.

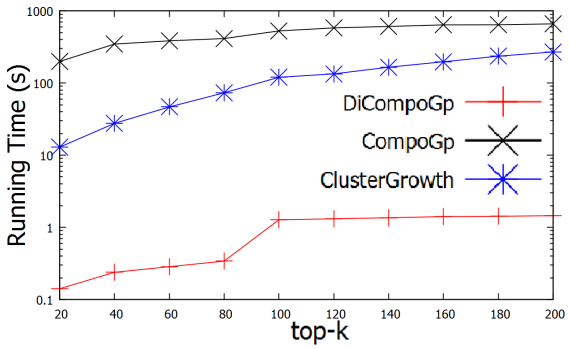
Figure 6.5 shows the compression gain of different algorithms. We can consider that DICOMPOGP, COMPOGP are significantly better than CLUSTERGROWTH. Another important property is that COMPOGP algorithm gives highly compressing patterns when K is small. However, with larger K it is clear that DICOMPOGP is more effective and outperforms the other algorithms. The reason is that by considering largest clusters which have not been used for data covering before, DICOMPOGP tends to choose patterns which do not overlap with the other ones in the dictionary $\mathcal{P}$. Therefore, new patterns which will be added into $\mathcal{P}$ usually cover non-overlapping part of the data. Meanwhile, COMPOGP does not take into account this point. However, the largest cluster-based rGpatterns are not always the best candidates and therefore with small K, COMPOGP is more effective.

Running time

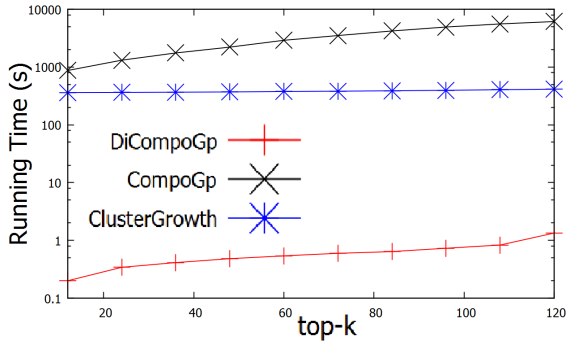
Figure 5.16 shows that the DICOMPOGP is the most efficient algorithm meanwhile COMPOGP is the worst. This is because: 1) COMPOGP is a post-processing algorithm which



(a) Swainsoni dataset.



(b) Buffalo dataset.



(c) Synthetic dataset.

Figure 5.16: Running time of different algorithms.

Table 5.5: The number of all the rGpatterns in datasets.

Dataset	**Swainsoni**	**Buffalo**	**Synthetic**
#rGpatterns	207	784	91,114

needs extra time to obtain top-K representative rGpatterns. 2) DICOMPOGP is better than CLUSTERGROWTH since it only needs to extract K, i.e. K is usually small, rGpatterns instead of a large number of all the candidates (see Table 5.5). 3) the DICOMPOGP algorithm computes the candidate compression gain without scanning the database.

Additionally, Figure 5.17 shows that the differences between rGpattern generation task by applying CLUSTERGROWTH, mining top-K representative patterns from all the candidates and DICOMPOGP are significant. Although CLUSTERGROWTH is known as a very efficient algorithm [10], it is wasteful to extract all the rGpatterns and then select top-K informative ones from them.

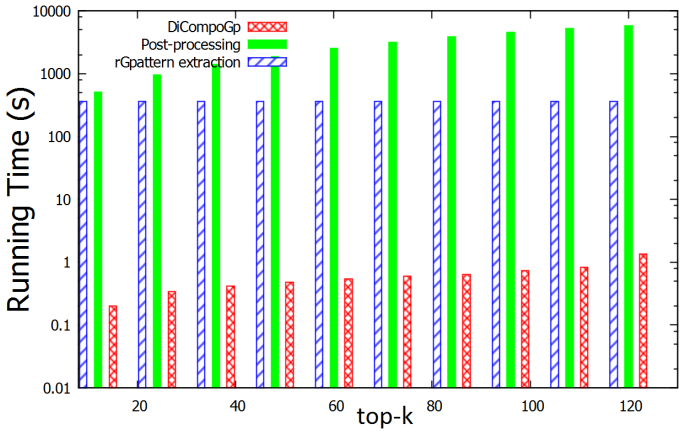
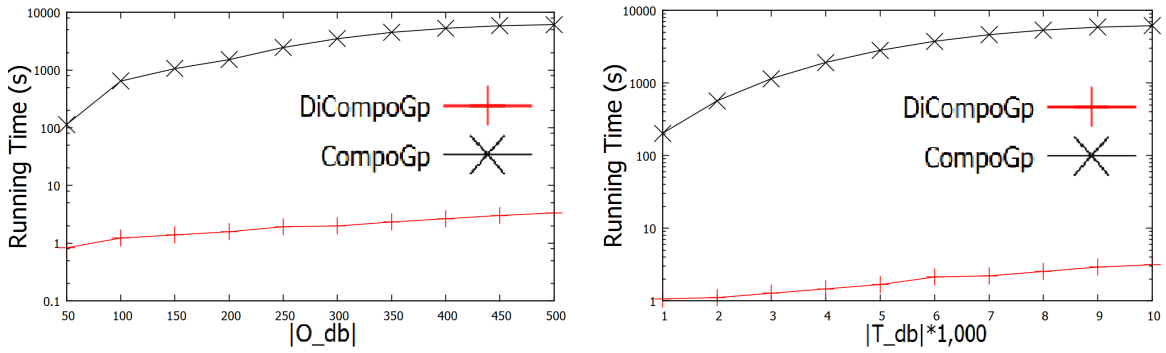


Figure 5.17: rGpattern generation task vs post-processing task vs DiCompoGp on Synthetic dataset.



(a) $|O_{db}|$. (b) $|T_{db}|$.

Figure 5.18: The DiCompoGp scalability on Synthetic dataset.

Scalability

In order to show the scalability of the DiCompoGp algorithm, we compare CompoGp and DiCompoGp by varying the size, i.e. $|O_{db}|$ and $|T_{db}|$, of the synthetic dataset. Figure 5.18 shows that the DiCompoGp is almost linear in terms of execution time. Another important point is that CompoGp is more sensitive than DiCompoGp. This is because CompoGp has to scan the database to compute the compression gain for each candidate at each iteration. Therefore, when the data size increases, it takes more computation time. Meanwhile, DiCompoGp can avoid the database scanning by applying Properties 16 and 17 and thus it is less sensitive.

5.7 Discussion

In this chapter, we propose the concepts of rGpattern and interesting maximal rGpattern. 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 rGpatterns. The proposed algorithm effectiveness, efficiency, parameter sensitiveness as well as the pattern meaning are demonstrated using real and large synthetic datasets.

This chapter opens up several avenues for short time research. One of the future directions is to apply maximal rGpattern in different applications. For instance, in bibliographic network analysis, ClusterGrowth algorithm can be exploited to analyze temporal evolution of research communities regarding specific topics. In this direction, an example of patterns could be *"from 2010 to 2012, the more time passes, the more papers are published in big data analysis field"*.

Another direction is to categorize maximal rGpatterns into different representative pattern groups. Naturally, the number of maximal rGpatterns properly is large and therefore it can be time consuming for analysts to exploit the moving object behaviors. Meanwhile, many patterns could be similar to each other since they share common objects and routes. The open issue here is to provide maximal rGpattern similarity (resp. dissimilarity) definition so that we can report distinct groups of similar patterns to end-users.

Nevertheless, after the extraction, the end user can be overwhelmed by a huge number of different movement patterns, i.e. convoys, closed swarms, group patterns, rGpatterns, 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 [31], using the minimum description length (MDL) principle [6], proposes to reduce the amount of itemsets by using an efficient encoding and then provide the end-user only with a set of informative patterns. However, they only work well with individual kind of patterns, i.e. itemset. Meanwhile there are different movement patterns and thus considering only specific kind of patterns results in not optimal compression patterns. Additionally, they can be overlap each other as well so enforcing non-overlapping patterns may cause losing meaningful patterns.

All of these challenging issues are expressed in the next chapter, *"Mining Representative Movement Patterns through Compression"*.

CHAPTER

6 Mining Representative Movement Patterns through Compression

Preamble

As we have seen in the previous chapter, 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.

6.1 Introduction

In this chapter, 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

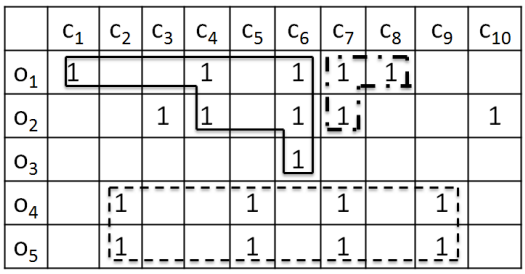


Figure 6.1: An example of moving object database. Shapes are movement patterns, o_i, c_i respectively are objects and clusters.

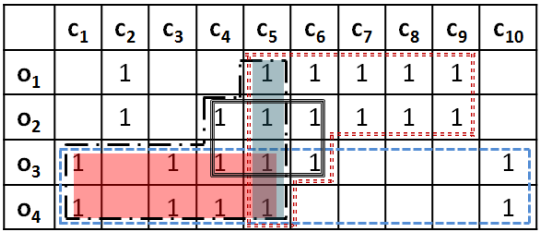


Figure 6.2: An example of pattern overlapping, between closed swarms (rectangles) and rGpatterns (step shapes), in Figure 1.1. Overlapping clusters are c_1, c_3, c_4, c_5 and c_6.

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 6.1, 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 6.2, there are four overlapping patterns. Unfortunately, Krimp algorithm does not allow overlapping patterns and thus 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.

The remaining sections of the chapter are organized as follows. Problem statement will be presented in Section 2. Encoding scheme and mining compression movement pattern algorithms will be respectively presented in Sections 3 and 4. Sections 5 and 6 show our comprehensive experiments and discussion.

6.2 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 22. *(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 chapter, the hypothesis is considered as a dictionary of movement patterns $\mathcal{P}$. Furthermore, as in [21], 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 23. *(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:*

$$\mathcal{P}^* = \arg\min_{\mathcal{P}} (L^*_{\mathcal{P}}(O_{db})) = \arg\min_{\mathcal{P}} (L^*(\mathcal{P}) + L^*(O_{db}|\mathcal{P})), \mathcal{P}^* \subseteq F \tag{6.1}$$

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.

Table 6.1: An illustrative example of database and dictionary in Figure 6.2. $\bar{0}$, $\bar{1}$ and $\bar{2}$ respectively are pattern types: closed swarm, $\mathrm{rGpattern}^{\geq}$ and $\mathrm{rGpattern}^{\leq}$.

O_{db}	Encoded O_{db}	Dictionary $\mathcal{P}$
$o_1 = c_2c_5c_6c_7c_8c_9$	$o_1 = c_2[p_2,4]$	
$o_2 = c_2c_4c_5c_6c_7c_8c_9$	$o_2 = c_2c_4[p_2,4]$	$p_1 = c_1c_3c_4c_5, \bar{1}$
$o_3 = c_1c_3c_4c_5c_6c_{10}$	$o_3 = p_3c_6$	$p_2 = [p_1,3]c_6c_7c_8c_9, \bar{2}$
$o_4 = c_1c_3c_4c_5c_{10}$	$o_4 = p_3$	$p_3 = [p_1,0]c_{10}, \bar{0}$

6.3 Encoding Scheme

6.3.1 Movement Pattern Dictionary-based Encoding

Before discussing our encoding for moving object data, we revisit the encoding scheme used in the Krimp algorithm [31]. 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.

In this chapter 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 6.1, we can consider that o_2 joins the $\mathrm{rGpattern}^{\geq} = c_1c_4c_6$ at c_4c_6. While, the closed swarm $cs = c_2c_5c_7c_9$ occurs in o_4 and o_5 entirely.

Property 21. *(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:*

$$\begin{cases} (1): p \textit{ is a } \mathrm{rGpattern}^{\geq}, \exists i \in [1,n] | \forall j \geq i, c_j \in o. \\ (2): p \textit{ is a } \mathrm{rGpattern}^{\leq}, \exists i \in [1,n] | \forall j \leq i, c_j \in o. \\ (3): p \textit{ is a closed swarm}, \forall j \in [1,n], c_j \in o. \end{cases} \tag{6.2}$$

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 $\forall j \in [1,n], c_j \in o$. □

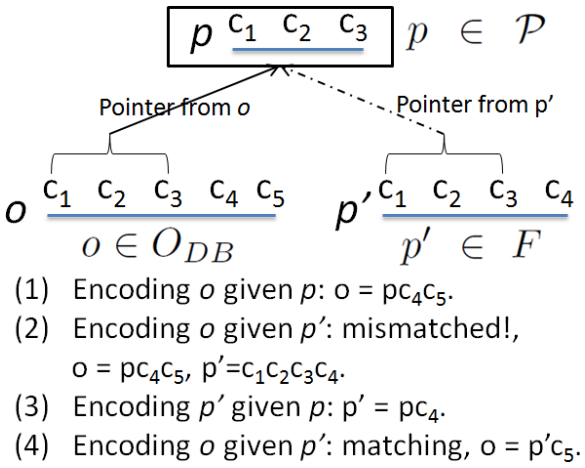


Figure 6.3: An example of the approach.

For instance, see Table 6.1, 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 21, 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. $\mathrm{rGpattern}^{\geq}$, $\mathrm{rGpattern}^{\leq}$), 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 $\mathrm{rGpattern}^{\geq}$ (resp. $\mathrm{rGpattern}^{\leq}$).

6.3.2 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 $\mathcal{P}$.

See Figure 6.3, 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

Table 6.2: Correlations between pattern p and pattern p' in F. O, Δ and X respectively mean *"overlapping allowed, regular encoding"*, *"overlapping allowed, no encoding"* and *"overlapping not allowed"*.

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

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 6.2. 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.

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 6.2, the objects o_1 and o_4 do not contribute to the closed swarm $p = c_4c_5c_6$. Thus, if the gradual trajectory $p' = c_1c_3c_4c_5$ is encoded given p to indicate the overlapping clusters c_4c_5 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 6.3.1 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 6.2 and Table 6.1, given a gradual trajectory pattern $\text{rGpattern}^{\geq}$ $p = p_1 = c_1c_3c_4c_5$, a $\text{rGpattern}^{\leq}$ $p' = p_2 = c_5c_6c_7c_8c_9$, the object $o_2 = c_2c_4c_5c_6c_7c_8c_9$. We first encodes o_2 given p such that $o_2 = c_2[p,2]c_6c_7c_8c_9$. Then,

p' is encoded given p, i.e. $p' = [p,3]c_6c_7c_8c_9$. We can consider that the two pointers referring to p from o_2 (i.e. $[p,2]$) and from p' (i.e. $[p,3]$) are different and thus o_2 and p' are mismatched. Therefore, o_2 cannot be encoded given p' despite the fact that p' essentially occurs in o_2.

To deal with this issue, we simply recover uncommon clusters between the two pointers. For instance, to encode o_2 by using p', we first recover uncommon cluster such that $o_2 = c_2c_4[p,3]c_6c_7c_8c_9$. Note that $[p,2] = c_4[p,3]$. Since $p' = [p,3]c_6c_7c_8c_9$, o_2 is encoded given p' such that $o_2 = c_2c_4[p',4]$.

Definition 24. *(Uncommon Clusters for* $\mathtt{rGpattern}^{\geq}$*). Given a* $\mathtt{rGpattern}^{\geq}$, $p = c_1 \dots c_n$ *and two pointers refer to* p, $[p,k]$ *and* $[p,l]$ *with* $k \leq l$. $\mathtt{uncom}(p,k,l) = c_kc_{k+1}\dots c_{l-1}$ *is called an uncommon list of clusters between* $[p,k]$ *and* $[p,l]$. *Note that* $[p,k] = c_kc_{k+1}\dots c_{l-1}[p,l]$.

Similarly, we also have $\mathtt{uncom}(p,k,l)$ in the case p is a $\mathtt{rGpattern}^{\leq}$. 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 $\mathtt{rGpattern}^{\geq}$ and p' is a closed swarm.

Lemma 7. *Given a* $\mathtt{rGpattern}^{\geq}$, $p = c_1 \dots c_n$, *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:*

$$k \leq l \wedge o = x_o\ \mathtt{uncom}(p,k,l)[p,l]\ y_o \tag{6.3}$$

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 21). That suffers the assumption and thus we have $k \leq l$. Deal to the Definition 24, $[p,k] = \mathtt{uncom}(p,k,l)[p,l]$. Consequently, we have $o = x_o\ \mathtt{uncom}(p,k,l)[p,l]\ y_o$. □

By applying Lemma 7, we have $o = x_o\ \mathtt{uncom}(p,k,l)[p,l]\ y_o$ 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 $\overrightarrow{(p',o)} = \{([p_1,l_1],[p_1,k_1]),\dots,([p_n,l_n],[p_n,k_n])\}$ 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 7 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 6.1 is

calculated as $L(\mathcal{P}) = |p_1|+1+|p_2|+1+|p_3|+1+|\mathcal{P}| = 4+1+6+1+3+1+3 = 19$. Similarly, description length of o_2 is $L(o_2|\mathcal{P}) = 2+|[p_2,4]| = 4$.

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.

6.4 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.

6.4.1 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 $p' = \mathtt{argmin}_{p^* \in F}(L_{p^*}(O_{db}))$. The NAIVECOMPO is presented in Algorithm 9.

6.4.2 Smart Greedy Approach

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 $\mathtt{gain}(p',\mathcal{P}) = L_{\mathcal{P}}(O_{db}) - L_{\mathcal{P}\cup p'}(O_{db})$.

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

$$\mathtt{gain}(p',\mathcal{P}) = |O_{p'}| \times |p'| - (\sum_{o}^{O_{p'}} \sum_{i}^{\overrightarrow{(p',o)}} |l_i - k_i| + |p'| + |O_{p'}| + 2) \tag{6.4}$$

Algorithm 9: NaiveCompo

Input : Database O_{db}, set of patterns F, int K
Output: Compressing patterns $\mathcal{P}$

```
1  begin
2      P ⟵ ∅;
3      while |P| < K do
4          foreach p ∈ F do
5              O^d_db ⟵ O_db;
6              L*(O^d_db|p) ⟵ CompressionSize(O^d_db, p);
7          p* ⟵ argmin_p L*(O^d_db|p);
8          P ⟵ p*; F ⟵ F \ {p*};
9          Replace all instances of p* in O_db by its pointers;
10         Replace all instances of p* in F by its pointers;
11     output P;
12 CompressionSize(O^d_db, p)
13 begin
14     size ⟵ 0;
15     foreach o ∈ O_db do
16         if p.involved(o) = true then
17             Replace instance of p in o by its pointers;
18     foreach o ∈ O_db do
19         size ⟵ size + |o|;
20     size ⟵ size + |p| + 1;
21     output size;
```

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 $L_{\mathcal{P}\cup p'}(O_{db}) = L(\mathcal{P}) + L(\overline{O}_{p'}|\mathcal{P}) + L(O_{p'}|\mathcal{P}) - |O_{p'}| \times |p'| + (\sum_o^{O_{p'}} \sum_i^{\overrightarrow{(p',o)}} |l_i - k_i| + |p'| + |O_{p'}| + 2)$. Note that $L(O_{db}|\mathcal{P}) = L(\overline{O}_{p'}|\mathcal{P}) + L(O_{p'}|\mathcal{P})$. Consequently, we have $gain(p', \mathcal{P}) = |O_{p'}| \times |p'| - (\sum_o^{O_{p'}} \sum_i^{\overrightarrow{(p',o)}} |l_i - k_i| + |p'| + |O_{p'}| + 2)$. □

By applying Property 22, we can compute the compression gain when adding a new closed swarm p' into the dictionary $\mathcal{P}$. In the Equation 6.4, the compression $gain(p', \mathcal{P})$

Algorithm 10: SmartCompo

Input : Database O_{db}, set of patterns F, int K
Output: Compressing patterns $\mathcal{P}$

```
1  begin
2      P ⟵ ∅;
3      while |P| < K do
4          foreach p ∈ F do
5              L*(O_db|p) ⟵ Benefit(O_db, p);
6          p* ⟵ argmin_p L*(O_db|p);
7          P ⟵ p*; F ⟵ F \ {p*};
8          Replace all instances of p* in O_db by its pointers;
9          Replace all instances of p* in F by its pointers;
10     output P;
11 Benefit(O_db^d, p)
12 begin
13     b ⟵ 0;
14     foreach o ∈ O_db do
15         if p.involved(o) = true then
16             b ⟵ b + benefit(o, p);
17     b ⟵ b + |p| + 1;
18     output b;
```

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 22.

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 10.

6.5 Experimental Results

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 [23] [10], 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

1. http://www.movebank.org

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

Figure 6.4: Top-3 typical compression patterns.

tracking time from year 2000 to year 2006. The original data has 26610 reported locations and 3001 timestamps. Similarly to [18] [23], we first use linear interpolation to fill in the missing data. Furthermore, DBScan [4] ($MinPts = 2; Eps = 0.001$) 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* [23] and the set of gradual trajectories extracted by *ClusterGrowth* [10].

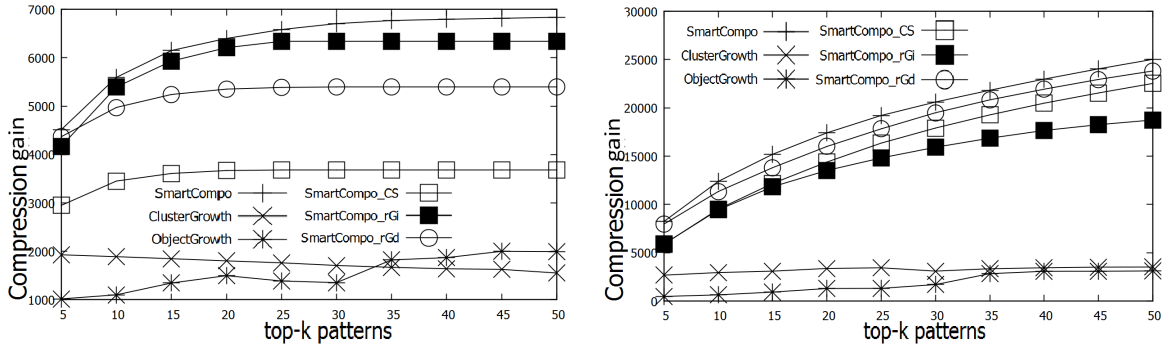
Effectiveness. We compare the top-5 highest support closed swarms, the top-5 highest covered area gradual trajectory patterns and the top-5 compression patterns from Swainsoni dataset. Each color represents a Swainsoni trajectory involved in the pattern.

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 6.4 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 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



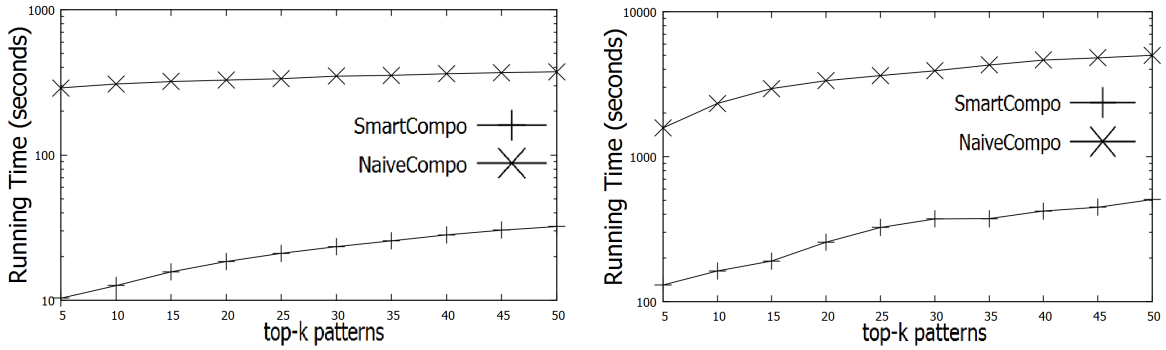
(a) Swainsoni dataset (b) Buffalo dataset

Figure 6.5: Compressibility (higher is better) of different algorithms.

individual pattern type, compression gain is calculated according to the greedy encoding scheme used for SmartCompo. They are respectively denoted as $\mathtt{SmartCompo\_CS}$ (i.e. for closed swarms), $\mathtt{SmartCompo\_rGi}$ (i.e. for $\mathtt{rGpattern}^{\geq}$) and $\mathtt{SmartCompo\_rGd}$ (i.e. for $\mathtt{rGpattern}^{\leq}$). 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 6.5 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.

One of the most interesting phenomena is that the Swainsonies and Buffaloes have quite different movement behavior. See Figure 6.5a, we can consider that $\mathtt{rGpattern}^{\geq}$ 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 6.5b, it clear that the compression gains of closed swarms, $\mathtt{rGpattern}^{\geq}$ and $\mathtt{rGpattern}^{\leq}$ 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.



(a) Swainsoni dataset (b) Buffalo dataset

Figure 6.6: Running time.

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 6.6). By exploiting the properties, we can directly select the best candidate at each iteration. Consequently, the process efficiency is speed up.

6.6 Discussion

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.

Even if SmartCompo is an effective algorithm, it is a straightforward approach and we need to handle the efficiency issue. Indeed, with dense and large datasets, the number of generated patterns is exponential and thus efficiency issue will be a key challenge. In this report we will address this issue by providing only a potential solution without experimental results in the Perspective Chapter.

As an extra work, we also present the notion of *multi-relational gradual patterns*. Gradual patterns highlight co-variations of attributes of the form *"The more/less X, the more/less Y"*. Existing techniques require all the interesting data to be in a single database relation or table. In the next chapter, we will extends the notion of gradual pattern to the case in which the co-variations are possibly expressed between attributes of different database relations.

CHAPTER

7 Mining Multi-Relational Gradual Patterns

Preamble

In this chapter, we will extends the notion of gradual pattern expressed between attributes of different database relations. Even this work do not directly relate to trajectory mining problem, it provide a way to understand the correlation between behaviors in a graduality point of view.

Gradual patterns highlight co-variations of attributes of the form "The more/less X, the more/less Y". *Their usefulness in several applications has recently stimulated the synthesis of several algorithms for their automated discovery from large datasets. However, existing techniques require all the interesting data to be in a single database relation or table. This chapter extends the notion of gradual pattern to the case in which the co-variations are possibly expressed between attributes of different database relations. The interestingness measure for this class of "relational gradual patterns" is defined through two different criteria: Kendall's* τ *and gradual support. Moreover, this chapter proposes two algorithms, named* τRGp *and* gRGp*, for the discovery of relational gradual rules, and introducing efficient pruning strategies to reduce the search space. The efficiency of the algorithms is empirically validated, and the quality of relational gradual patterns is proved on real-world databases.*

7.1 Introduction

Nowadays, most of information systems are based on the relational database technology. The logical models of the data are sets of relations or tables possibly linked by foreign key constraints. This contrasts with the usual practice in Data Mining of organizing data in

a single relation when they are analyzed. Relational data mining approaches [45] are characterized by both their direct applicability to “multi-relational data" (MRD), i.e., standard relational databases, and their capability of looking for patterns which involve multiple relations.

Most of the studies on relational data mining focus on relational patterns at the tuple level, i.e., they express relationships between tuples of different database relations. Relational association rules [43], relational naïve Bayesian classifiers [46], relational regression models [40] and relational subgroups [56], all express patterns as either SQL queries or first-order logic clauses with constraints between tuples or facts. Similarly, the probabilistic relational models [47] define a distribution over a set of instances of a schema, and consider the structure at the level of attribute values.

In this chapter, we focus on patterns expressing the relational structure at the attribute level. In particular, we consider relational extensions of the class of gradual dependencies, which express covariations of attributes of the form *“The more/less X, the more/less Y"* [49]. This class of patterns has a wide range of applications. For instance, with reference to the Financial database in Figure 7.1, the gradual pattern *“the higher the average salary in a district, the bigger the deals made by inhabitants living in the district"* could be useful for business planning, while the gradual pattern *“the smaller the district, the longer the duration of the loans"* provides financial promoters with some useful insights. Interestingly, these two examples show gradual dependencies between attributes from different relations (the average salary, the size of districts, the deal and the loan), hence they could be discovered only in a MRD setting.

To discover this kind of patterns, we first introduce the concept of relational gradual pattern (*mrGP*), and its associated support measures based on *Kendall's* τ [42] and *gradual support* [44]. Then we propose an algorithm, called RGp, to discover mrGPs directly from MRD. Mining the complete set of mrGPs is a non-trivial task since the size of the search space is exponential in the number of numerical attributes of the multi-relational database. To tame computational complexity, we introduce three efficient pruning strategies named *Complementary Pruning*, *Apriori Pruning* and *Backward Pruning* to reduce the search space and prevent unnecessary computation. RGp has two instances, named τRGp and gRGp. Experiments conducted on real datasets demonstrate the effectiveness, pattern meaning and efficiency of our methods.

The remain of this chapter is organized as follows. I will give the preliminarily definitions of relational data, multi-relational gradual pattern in Section 2. Pattern occurrence and pattern support will be introduced in Sections 3 and 4. Section 5 focuses on proposing an efficient algorithm to extract multi-relational gradual patterns. Experimental results will be conducted in Section 6 and Section 7 is our conclusion.

7.2 Preliminarily Definitions

In this section we formalize the data model, the notion of relational gradual pattern, and its associated interestingness measure.

7.2.1 Multi-Relational Data

In this work, we assume that a database $\mathtt{db}$ consists of a set of tables, $\mathtt{db} = \{T^1, \ldots, T^n\}$, each of which has a schema $S(T^i) = (PK^i, \mathcal{FK}^i, \mathcal{A}^i)$ consisting of a *primary key* (or simply *key*) PK^i, possibly some *foreign keys* (set $\mathcal{FK}^i$), and at least one *attribute* (set $\mathcal{A}^i$). For instance, with reference to Figure 7.1, the Loan table has a key loan-ID, a foreign key *account-ID* and attributes *date, amount, duration, payment, status*.

Foreign keys define the only possible joins between two tables. Without loss of generality, we assume that any pair of tables have at most one foreign key linking them. Indeed, databases can always be losslessly recoded such that this assumption holds.

Moreover, we assume that all attributes have a numerical domain, since we are not interested in finding patterns among categorical attributes. The numerical domain of the attribute A^i_j is denoted as $\mathrm{Dom}(A^i_j)$. Similarly, the domain of table T^i is denoted by $\mathrm{Dom}(T^i)$, and corresponds to the Cartesian product of all domains involved, i.e.,

$$\mathrm{Dom}(T^i) = \mathrm{Dom}(PK^i) \times \prod_{\mathcal{FK}^i_j \in \mathcal{FK}^i} \mathrm{Dom}(\mathcal{FK}^i_j) \times \prod_{A^i_j \in \mathcal{A}^i} \mathrm{Dom}(A^i_j) \tag{7.1}$$

Each table is also associated with a set of tuples, the *table extension*. Henceforth, the distinction between the table and its extension is blurred, and we use the notation $t \in T^i$ to indicate that a tuple t is in extension of table T^i.

The database as whole should satisfy *referential integrity*, i.e., foreign-key values in a tuple refer to existing tuples in the table for which this foreign key refers to. Formally, if $t \in T^i$ and $S(T^i) = (PK^i, \mathcal{FK}^i, \mathcal{A}^i)$, then for each $FK^i_j \in \mathcal{FK}^i$ whose domain is $\mathrm{Dom}(PK^l)$, the following condition must hold: there exists a tuple $t' \in T^l$ such that $\pi_{FK^i_j}(t) = \pi_{PK^l}(t')$, where π is the usual projection operator of relational algebra.

7.2.2 Gradual Pattern: Single Relation vs Multi-Relations

Mining gradual pattern from single relation data has been studied for a long time [44] [42] [57]. However, existing approaches cannot be directly applied on multi-relational data context. One of potential solutions is to join the relations to be a single table and then traditional approaches such as GRITE [44] and proposed algorithms in [42] can be applied to extract gradual patterns. This simplicity has some drawbacks.

First of all, joining all the relations will generate a huge amounts of data even for small dataset. The number of transaction after joining all the relations in the database

$db = \{T^1, \ldots, T^n\}$ can reach to $\prod_i^n |T^i|$ where $|T^i|$ is the number of transactions in relation T^i. For instance, see Financial database illustrated in Figure 7.1, there could be 10^{26} transactions totally in this case. It rapidly becomes inefficient for existing gradual pattern mining algorithms.

Second, the total number of pattern candidates is $2^{2\sum_{i=1}^{n}|\mathcal{A}^i|}$ where $|\mathcal{A}^i|$ is the number of attributes in $\mathcal{A}^i$. Computing such large number of candidates from such huge single-relational database quickly becomes infeasible for larger databases or candidate collections. Moreover, as many patterns which have support values lower than support threshold will be ignored, joining all the relations together is quite wasteful.

Another important issue is that joining all the relations will lead to an incorrect meaning of pattern support. Indeed, each tuple is considered as an object in single relational data and therefore all possible pairs of tuples will be taken into account in pattern support definition [44] [42] [57]. However, different tuples in the joining table can represent the same object since they can have the same primary key that results in a set of confusing gradual patterns. This issue will be clearly explained in Section 4: Pattern Support.

In this chapter, we address these issues. First of all we generalize gradual pattern definition from single relation data to multi-relational data. Then we further design an efficient algorithm to directly extract the complete set of all gradual patterns from a multi-relational dataset.

7.2.3 Multi-Relational Gradual Pattern

In our context, an increasing attribute A_j^i is denoted as $A_j^{i>}$ and a decreasing attribute is denoted as $A_j^{i<}$. Notice that $A_j^{i<}$ and $A_j^{i>}$ complement each other. Let E^i is the set of all single gradual attributes which are generated by $\mathcal{A}^i$ from table T^i, we have:

$$E^i = \bigcup_{j=1}^{|\mathcal{A}^i|} \{A_j^{i<}\} \cup \{A_j^{i>}\} \tag{7.2}$$

Naturally, in a single table, a mono relational gradual pattern q^i is a set of gradual attributes such that $q^i = \{A_1^{i>}, A_2^{i<}, \ldots, A_m^{i>}\}$ where $A_1^{i>}, A_2^{i<}, \ldots, A_m^{i>} \in E^i$. The set of all potential gradual patterns Q^i contains all the possible gradual patterns which are generated by combining the elements in E^i. Furthermore, *"proper"* relational gradual patterns must cross multiple tables and perhaps without considering some mono relational gradual pattern at some tables.

For instance, see Figure 7.1, a potential pattern could be $\{District.avg\text{-}salary^{>}, Loan.amount^{<}\}$ which does not contain any mono relational gradual patterns from Account table.

To express this phenomenon, we propose a crossing element, denoted as Ω, for each relation $T^i \in db$ and of course $\Omega \in Q^i$. The pattern $\{District.avg\text{-}$

$\mathtt{salary}^{>}$, $\mathtt{Account.\Omega}, \mathtt{Loan.amount}^{<}\}$ can be presented as $\{\mathtt{District.avg\text{-}salary}^{>}$, $\mathtt{Loan.amount}^{<}\}$.

Definition 25. *(Relational Gradual Pattern). Let* $\mathtt{db} = \{T^1, \ldots, T^n\}$ *be a database for which each table* T^i *has a schema* $S(T^i) = (PK^i, \mathcal{FK}^i, \mathcal{A}^i)$.

1. *Mono relational gradual pattern (*$\mathtt{mono\text{-}rGp}$*).* $p = q^i$ *where* $q^i \in Q^i$.
2. *Relational gradual pattern between two relations.*

a. 1:N relation. $p = q^i q^j$ *where* $q^i \in Q^i$, $q^j \in Q^j$ *and* $PK^i \subseteq \mathcal{FK}^j$.

b. M:N relation. $p = q^i q^j$ *where* $q^i \in Q^i$, $q^j \in Q^j$ *and* $\exists T^k \in \mathtt{db}$ *s.t.* $(PK^i \subseteq PK^k$ *and* $PK^j \subseteq PK^k)$ *or* $(PK^i \subseteq \mathcal{FK}^k$ *and* $PK^j \subseteq \mathcal{FK}^k)$.

3. *Multi-Relational gradual pattern (rGp).* $p = q^1 q^2 \ldots q^m$ *where* $\forall i \in \{1, \ldots, m-1\}$: $q^i \in Q^i$ *and* $q^i q^{i+1}$ *verifies the second case. Note that* q^1 *can be extracted from any relation* $T \in \mathtt{db}$.

In the Definition 25, we define the patterns that can be extracted in a single relation, in two relations as well as in multi-relations. In the first case, the patterns p are extracted from a single relation T^i. Then, the combination of two patterns from two relations is the second case. The relations from T^i to T^j in *2.a* and *2.b* respectively are 1:N and M:N relationships. Note that, in *2.b*, the table T^k is a connection relation between T^i and T^j. Finally, the third case is a generalization of the second case so that the pattern p is defined as a mrGp in which all pairs of consecutive mono-rGps verifies the second case. For instance, see Figures 7.1, potential interesting mrGps are as follows.

1:N example. In table Loan, we have a mono-rGp such as $q^2 = \{\mathtt{amount}^{>}, \mathtt{duration}^{>}\}$. Additionally, we can have a larger pattern by connecting Loan table to Account table over foreign key $\mathtt{account}_{ID}$ (i.e. $\mathtt{Account.PK} = \mathtt{Loan.\mathcal{FK}}$). The pattern candidate can be $p = q^1 q^2 = \{\mathtt{Account.frequency}^{>}, \mathtt{Loan.amount}^{>}, \mathtt{Loan.duration}^{>}\}$. The meaning is *"the more frequently the accounts are used, the more amount and the longer duration loans are"*.

M:N example. Another example is $p = \{\mathtt{Client.birthdate}^{<}, \mathtt{Account.frequency}^{>}\}$ since $(PK^{Client} \subseteq \mathcal{FK}^{Disposition}$ and $PK^{Account} \subseteq \mathcal{FK}^{Disposition})$, the meaning is *"the older clients are, the more frequently they use their accounts"*.

mrGp. A potential mrGp is $p = \{\mathtt{District.\#people}^{>}$, $\mathtt{Account.frequency}^{>}$, $\mathtt{Loan.amount}^{>}\}$. The meaning is *"the more number of inhabitants the districts have, for the habitants living in the districts, the more frequently their accounts are used and the more amount of loans they have"*.

In common, relational patterns are defined as alphabet patterns that select one attribute and assign it one value [50] [54] [55]. However, in relational gradual pattern, we are working at attribute level. Therefore, the pattern domain is quite different from the common alphabet pattern domain.

Definition 26. *(Gradual Pattern Domain). Given a set of gradual attributes* $s = \{A_1^*, A_2^*, \ldots, A_n^*\}$ *where* $\forall A_i^* \in s$, A_i^* *can belong to any relation in* db. *The domain of a*

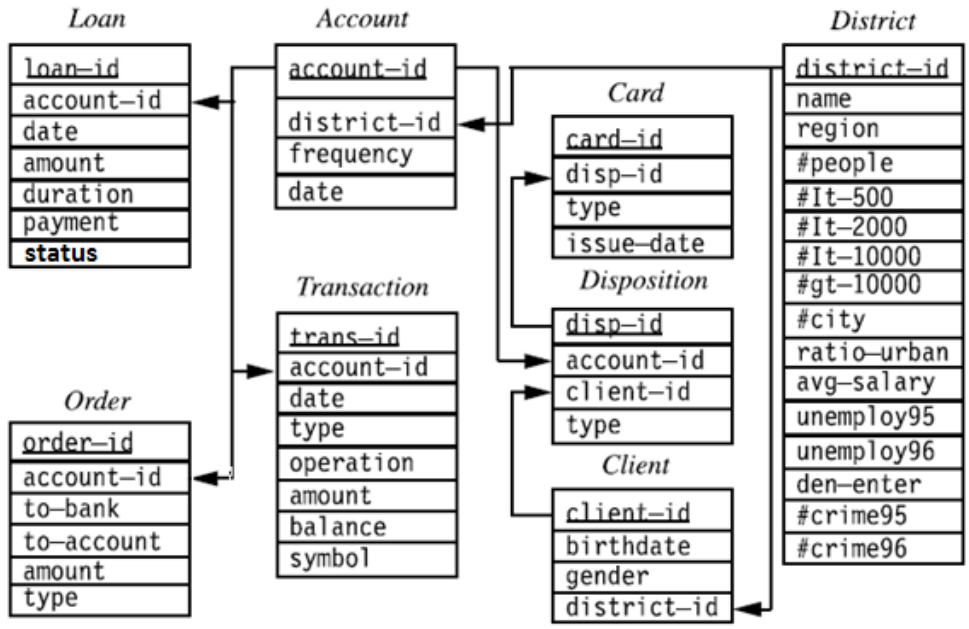


Figure 7.1: A Financial database (from PKDD CUP 99).

pattern p *which is formed by combining the elements in* s*, denoted* $\mathrm{Dom}(p)$*, is defined as follows.*

$$\forall A_i^* \in s, \mathrm{Dom}(A_i^*) = \{A_i^{*>}, A_i^{*<}\}. \mathrm{Dom}(p) = \prod_{A_i^* \in s} \mathrm{Dom}(A_i^*) \tag{7.3}$$

7.3 Pattern Occurrences

The common relational patterns usually are rather complicated structures. For instance, a pattern can contain a set of tuples (e.g. as in [54]) or a set of sets of sets of ... of tuples (e.g. as in [50]). However, our patterns can become more complicated, since it requires to consider at least two tuples at the same time to evaluate the graduality of an attribute. Thus, a pair of tuples is used to examine the graduality of a set of attributes. The gradual tuple pair can be defined as follows:

Definition 27. *(Gradual Tuple Pair). Given a gradual variant* $\circledast \in \{<, >\}$ *and a mono-rGp* $q^i = \{A_1^{i\,\circledast}, \ldots, A_m^{i\,\circledast}\} \in Q^i$*. A pair of distinct tuples* (t_x, t_y) *s.t.* $t_x, t_y \in T^i$*, is gradual in respect to* q^i *if:*

$$\forall l \in \{1, \ldots, m\}, \pi_{A_l^i}(t_x) \circledast \pi_{A_l^i}(t_y) \tag{7.4}$$

Given a mrGp $p = q^1 q^2 \dots q^m$*, a pair of tuples* (t_x, t_y) *s.t.* $t_x, t_y \in (T^1 \bowtie T^2 \bowtie \dots \bowtie T^m)$*, is gradual in respect to* p *if:*

$$\forall l \in \{1, \dots, m\}, (\pi_{T^l}(t_1), \pi_{T^l}(t_2)) \textit{ respects } q^l \tag{7.5}$$

Definition 28. *(Crossing Element Occurrence). Given a crossing element* $\Omega \in Q^i$*,* $\forall (t_x, t_y)$ *where* $t_x, t_y \in T^i$*,* $(t_x, t_y) \in occ(\Omega)$*.*

Definition 29. *(Pattern Occurrence). Let* $db = \{T^1, \dots, T^n\}$ *be a database for which each table* T^i *has a schema* $S(T^i) = (PK^i, \mathcal{FK}^i, \mathcal{A}^i)$*. Given a mrGp* $p = q^1 q^2 \dots q^m$ *where* $\forall i \in [1, m] : q^i \in T^i$*, the occurrence of* p*, denoted* $occ(p)$*, is a set of gradual tuple pairs* (t_x, t_y) *in respect to* p*. Note that* $t_x, t_y \in (T^1 \bowtie T^2 \bowtie \dots \bowtie T^m)$*.*

For brevity, we only report primary key and foreign keys in a pattern occurrence. For instance, see Figure 7.2, let us assume that the pattern $q^1 = \{Account.frequency^{>}\}$ and one of the occurrences of q^1 is (t_1, t_2). Similarly, we also have the occurrences of $q^2 = \{Loan.amount^{>}\}$. Moreover, if we have the pattern $p = q^1 q^2$ then an occurrence of p is $((t_1, t_2), (t'_1, t'_3))$ and $\pi_{account_{ID}}(t_1) = \pi_{account_{ID}}(t'_1) = 10, \pi_{account_{ID}}(t_2) = \pi_{account_{ID}}(t'_3) = 11$. Obviously, we can compute the occurrences of p as illustrated in Figure 7.2.

A relational gradual pattern can thus contain sets of gradual tuple pairs. Hence, it is illustrative to consider what the domain of such patterns is. That is, *"what does an instance look like?"* Essentially, the definition of these domains follows the structure of the pattern occurrence.

Definition 30. *(Pattern Occurrence Domain). Let* $db = \{T^1, \dots, T^n\}$ *be a database for which each table* T^i *has a schema* $S(T^i) = (PK^i, \mathcal{FK}^i, \mathcal{A}^i)$*. Furthermore, let* $p = q^1 q^2 \dots q^m$ *be a mrGp. The domain of* p *occurrences, denoted* $DomOcc(p)$ *is given by*

$$DomOcc(p) = (Dom(T^1 \bowtie T^2 \bowtie \dots \bowtie T^m), Dom(T^1 \bowtie T^2 \bowtie \dots \bowtie T^m)) \tag{7.6}$$

7.4 Pattern Support

Generalization gradual pattern support definition from single relation to multi-relational data context is a non-trivial task. Naturally, traditional support of a gradual pattern [42] [44] is defined as the number of pattern occurrences over all possible cases, i.e. $supp(p) = \frac{|occ(p)|}{|\textit{all possible cases}|}$. That fact's that defining the $|\textit{all possible cases}|$ in a single table T^i is quite simple, i.e. equal to $\binom{|T^i|}{2}$ as in [42] or $|T^i|$ as in [44]. However, it is completely different and becomes a challenging issue in multi-relational data context.

Let us consider the example in Figure 7.2, to compute the support for the pattern $\{Account.frequency^{>}, Loan.amount^{>}\}$, one of potential solutions is to join two tables

	T^1 = Account		
	$account_{ID}$	frequency	date
t_1	10	2	06/2007
t_2	11	3	03/2006
t_3	12	4	06/2006
t_4	13	3	03/2008

	T^2 = Loan			
	$loan_{ID}$	$account_{ID}$	date	amount
t'_1	30	10	06/2008	10245
t'_2	31	10	09/2008	13722
t'_3	32	11	08/2006	25313
t'_4	33	12	09/2006	27147
t'_5	34	12	05/2008	27194

Gradual Tuple Pairs	**occ(Account.frequency$^>$)**	
	$account_{ID}$	**$account_{ID}$**
(t_1, t_2)	10	11
(t_1, t_3)	10	12
(t_2, t_3)	11	12
(t_1, t_4)	10	13
(t_4, t_3)	13	12

Gradual Tuple Pairs	**occ(Loan.amount$^>$)**			
	$loan_{ID}$	$account_{ID}$	**$loan_{ID}$**	$account_{ID}$
(t'_1, t'_2)	30	10	31	10
(t'_1, t'_3)	30	10	32	11
(t'_1, t'_4)	30	10	33	12
(t'_1, t'_5)	30	10	34	12
(t'_2, t'_3)	31	10	32	11
(t'_2, t'_4)	31	10	33	12
(t'_2, t'_5)	31	10	34	12
(t'_3, t'_4)	32	11	33	12
(t'_3, t'_5)	32	11	34	12
(t'_4, t'_5)	33	12	34	12

Figure 7.2: A 1:N relation example (best view in color).

	$loan_{ID}$	$account_{ID}$	amount	frequency
t''_1	30	10	10245	2
t''_2	31	10	13722	2
⋮	⋮	⋮	⋮	⋮
t''_a	33	12	27147	4
t''_b	34	12	27194	4

Figure 7.3: $\mathrm{Loan} \bowtie \mathrm{Account}$.

Account and Loan, i.e. see Figure 7.3, then apply existing algorithms [42] [44]. It is easy to consider that the tuple pairs (t''_1, t''_2) and (t''_a, t''_b) are still counted in $|\textit{all possible cases}|$ in existing algorithms [42] [44]. However, they should not been counted since they come from the same account which has a unique account frequency value, i.e. $\pi_{\text{account}_{\text{ID}}}(t''_1) = \pi_{\text{account}_{\text{ID}}}(t''_2) = 11$ and $\pi_{\text{account}_{\text{ID}}}(t''_a) = \pi_{\text{account}_{\text{ID}}}(t''_b) = 12$, so they never support the pattern. This is because the relationship between Account and Loan is 1:N. Furthermore, the $|\textit{all possible cases}|$ also depends on the relationship between two tables, i.e. 1:N or M:N. Thus, what we show here is that defining gradual pattern support definition in multi-relational data context is not straightforward and it demands an innovative definition.

Breakthrough these problems, in this chapter we propose novel support measures to effectively evaluate the graduality of a mrGp so that the pattern meaning is still remained.

7.4.1 *Kendall's* τ-based Multi-Relational Gradual Pattern Support

We start this section by reviewing the *Kendall's* τ-based support in [42]. Assume that A^i_1 and A^i_2 are two numerical attributes in table T^i. The measure can now be defined as follows:

$$\text{supp}_\tau(A^i_1, A^i_2) = \frac{|\{(t_x, t_y) \in (T^i)^2 | \pi_{A^i_1}(t_x) < \pi_{A^i_1}(t_y) \wedge \pi_{A^i_2}(t_x) < \pi_{A^i_2}(t_y)\}|}{\binom{|T^i|}{2}} \tag{7.7}$$

We can consider that $\binom{|T^i|}{2}$ is the $|\textit{all possible cases}|$ which contain whatever the tuple pair while all gradual tuple pairs (t_x, t_y) will be consider in the pattern occurrence. Now, let us define a set of operations that will be used in the multi-relational gradual pattern support definition.

– $\texttt{distinct}(PK^i, T^j)$: given two tables T^i and T^j so that $PK^i \subseteq \mathcal{FK}^j$ (1:N relationship). $\texttt{distinct}(PK^i, T^j)$ returns a set of distinct values $pk \in PK^i$ so that $\exists t' \in T^j : pk = \pi_{PK^i}(t')$. $\texttt{distinct}(PK^i, T^j) \longleftarrow$ *Select distinct* PK^i *from* T^j.

– $\texttt{count}(pk, T^j)$: given a primary key value $pk \in PK^i$, returns the number of tuples $t' \in T^j$ s.t. $pk = \pi_{PK^i}(t')$. We only consider pk having $\texttt{count}(pk, T^j) > 1$. $\texttt{count}(pk, T^j) \longleftarrow$ *Select count(*) from* T^j *where* PK^i *= pk having count(*) > 1.*

Definition 31. *(Kendall's τ support of mrGp). Let* $db = \{T^1, \ldots, T^n\}$ *be a database for which each table* T^i *has a schema* $S(T^i) = (PK^i, \mathcal{FK}^i, \mathcal{A}^i)$.

1. *Mono-rGp.* $p = q^i$.

$$\text{supp}_\tau(p) = \frac{|\text{occ}(p)|}{\binom{|T^i|}{2}} \tag{7.8}$$

2. *Relational gradual pattern between two relations.*

a. 1:N relation. $p = q^i q^j$ *where* $q^i \in Q^i$, $q^j \in Q^j$ *and* $PK^i \subseteq \mathcal{FK}^j$.

$$supp_\tau(p) = \frac{|occ(p)|}{\binom{|T^j|}{2} - \sum_{pk \in distinct(PK^i, T^j)} \binom{count(pk, T^j)}{2}} \tag{7.9}$$

b. M:N relation. $p = q^i q^j$ *where* $q^i \in Q^i$, $q^j \in Q^j$ *and* $\exists T^k \in db$ *s.t.* $(PK^i \subseteq PK^k$ *and* $PK^j \subseteq PK^k)$ *or* $(PK^i \subseteq \mathcal{FK}^k$ *and* $PK^j \subseteq \mathcal{FK}^k)$.

$$supp_\tau(p) = \frac{|occ(p)|}{\binom{|distinct(PK^i, T^k)|}{2} \times \binom{|distinct(PK^j, T^k)|}{2}} \tag{7.10}$$

c. Crossing element, Ω. $p = q^i \Omega$ *where* $q^i \in Q^i$, $\Omega \in Q^j$ *and* T^i, T^j *respects the cases 2a, 2b then we only need to transform instances* $(t_x, t_y) \in occ(p)$ *to* (t'_x, t'_y) *with* $t'_x, t'_y \in T^j$ *as follows:*
Case 2a: $occ(p) \longleftarrow$ *Select* $(T^j.t'_x, T^j.t'_y)$ *from* $occ(q^i), T^j$ *where* $\pi_{PK^i}(occ(q^i).t_x) = \pi_{PK^i}(T^j.t'_x)$ *and* $\pi_{PK^i}(occ(q^i).t_y) = \pi_{PK^i}(T^j.t'_y)$.
Case 2b: $occ(p) \longleftarrow$ *Select* $(T^j.t''_x, T^j.t''_y)$ *from* $occ(q^i), T^j, T^k$ *where* $(\pi_{PK^i}(occ(q^i).t_x) = \pi_{PK^i}(T^k.t'_x)$ *and* $\pi_{PK^i}(occ(q^i).t_y) = \pi_{PK^i}(T^k.t'_y))$ *and* $(\pi_{PK^j}(T^k.t'_x) = \pi_{PK^j}(T^j.t''_x)$ *and* $\pi_{PK^j}(T^k.t'_y) = \pi_{PK^j}(T^j.t''_y))$.
3. mrGp. $p = q^1 q^2 \dots q^{m-1} q^m$.

$$\begin{cases} (1): \textit{If } q^{m-1} q^m \textit{ respects the case 2a then} \\ supp_\tau(p) = \frac{|occ(p)|}{\binom{|T^m|}{2} - \sum_{pk \in distinct(PK^{m-1}, T^m)} \binom{count(pk, T^m)}{2}} \\ (2): \textit{If } q^{m-1} q^m \textit{ respects the case 2b then} \\ supp_\tau(p) = \frac{|occ(p)|}{\binom{|distinct(PK^{m-1}, T^k)|}{2} \times \binom{|distinct(PK^m, T^k)|}{2}} \end{cases} \tag{7.11}$$

Explanation. Regarding to a mono-rGp, we take all gradual tuple pairs (t_x, t_y) which support p, i.e. $occ(p)$. This number is then divided by the maximal possible number of such pairs which is $\binom{|T^i|}{2}$, and not $|T^i|(|T^i|-1)$ since it is not possible that both (t_x, t_y) and (t_y, t_x) support pattern p. For instance, see Figure 7.2, $p = \{Account.frequency^>\}$, $|occ(p)| = 5$ and $supp_\tau(p) = \frac{5}{6}$.
1:N relation. We can consider that the number of all possible gradual tuple pairs supports p is $\binom{|T^j|}{2}$. However, as mentioned before we need to ignore pairs of tuples which come from the same object, i.e. same primary key $pk \in PK^i$. Since they will never support p and the number of total of them is $\sum_{pk \in distinct(PK^i, T^j)} \binom{|count(pk, T^j)|}{2}$. Thus, $|$*all possible cases*$|$ in this situation is $\binom{|T^j|}{2} - \sum_{pk \in distinct(PK^i, T^j)} \binom{|count(pk, T^j)|}{2}$.

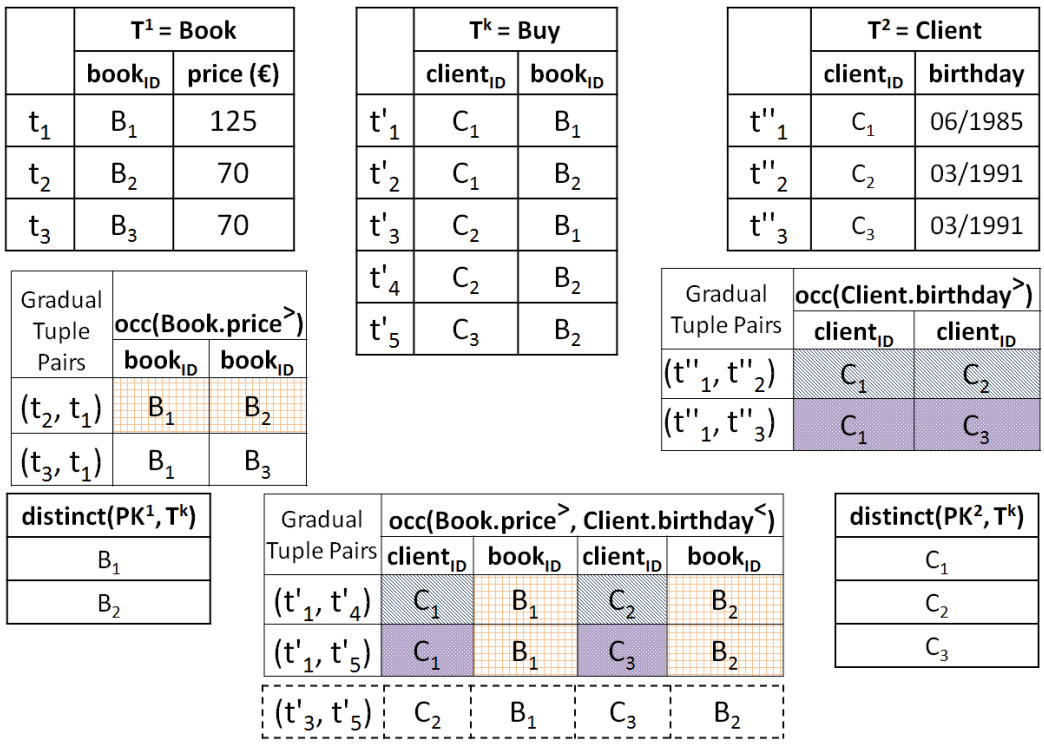


Figure 7.4: A M:N relation example. (t'_3, t'_5) does not support the pattern $p = \{Book.price^{>}, Client.birthday^{<}\}$ since $(C_2, C_3) \notin occ(Client.birthday^{<})$

.

For instance, see Figure 7.2, mrGp $p = q^1q^2 = \{Account.frequency^{>}, Loan.amount^{>}\}$. We have $|occ(q^1)| = 5, |occ(q^2)| = 10, |occ(p)| = 8$ and $distinct(PK^1, T^2) = \{10, 11, 12\}$, $count(10, T^j) = 2$, $count(11, T^j) = 1$, $count(12, T^j) = 2$. Thus, the *Kendall's* τ support of p is $supp_\tau(p) = \frac{8}{10-(\frac{2}{2})-(\frac{2}{2})} = 1$. Note that $count(11, T^j) = 1$ is not taken into account.

M:N relation. Essentially, identifying an instance $((t_x, t_y), (t'_x, t'_y)) \in occ(p)$ is to combine a pair of distinct keys $(pk_x, pk_y) : pk_x \neq pk_y, pk_x, pk_y \in Dom(PK^i)$ and another pair of distinct keys $(pk'_x, pk'_y) : pk'_x \neq pk'_y, pk'_x, pk'_y \in Dom(PK^j)$. Indeed, all the distinct primary key values PK^i which have relationship with some object in T^j are $distinct(PK^i, T^k)$. Thus the number of all possible combinations (pk_x, pk_y) is $\binom{|distinct(PK^i, T^k)|}{2}$. Similarly, we also have the number of all possible combinations (pk'_x, pk'_y) is $\binom{|distinct(PK^j, T^k)|}{2}$. Thus, $|$*all possible cases*$|$ in this situation is $\binom{|distinct(PK^i, T^k)|}{2} \times \binom{|distinct(PK^j, T^k)|}{2}$.

For instance, see Figure 7.4, a mrGp pattern $p = \{Book.price^{>}, Client.birthday^{<}\}$ is generated from two tables *Book* and *Client* with an M:N relationship via *Buy* table. In

Buy table, there are only two books B_1, B_2 are bought by clients and we need to ignore B_3 since it has not existed in *Buy*. So, $\mathtt{distinct}(PK^1, T^k) = \{B_1, B_2\}$. Similarly, we also have $\mathtt{distinct}(PK^2, T^k) = \{C_1, C_2, C_3\}$. Now we combine tuples (t'_x, t'_y) in $T^k = \mathtt{Buy}$ to check whether they support the pattern p. Indeed, $\pi_{\mathtt{client_{ID}}}(t'_x) \neq \pi_{\mathtt{client_{ID}}}(t'_y)$ and thus all the combinations of (t'_x, t'_y) in terms of $\mathtt{client_{ID}}$ can support p are $\binom{|\mathtt{distinct}(PK^2, T^k)|}{2} = 3$, i.e. $(C_1, C_2), (C_1, C_3)$ and (C_2, C_3). Similarly, all the combinations of (t'_x, t'_y) in terms of $\mathtt{book_{ID}}$ can support p are $\binom{|\mathtt{distinct}(PK^1, T^k)|}{2} = 1$. Gathering two attributes together, $|$*all possible cases*$|$ of (t'_x, t'_y) in terms of $\mathtt{client_{ID}}$ and $\mathtt{book_{ID}}$ which can support p are $\binom{|\mathtt{distinct}(PK^1, T^k)|}{2} \times \binom{|\mathtt{distinct}(PK^2, T^k)|}{2} = 1 \times 3 = 3$. Consequently, $\mathrm{supp}_\tau(p) = \frac{2}{3}$.

Crossing element Ω. We only need to transform the occurrences of p when Ω is added to p since Ω is used to pass the relation T^j. Then, we can add other mono-rGps $q^{j+1} \in Q^{j+1}$ to $p = q^i$ where T^{j+1} has a relation from T^j. The meaning of pattern $p = q^i \Omega q^{j+1}$ remains the same to $p = q^i q^{j+1}$. By applying crossing element Ω, we can extract patterns with combine attributes from indirect connected relations. Consequently, it supplies an insightful picture of the data structure in relational databases.

Multi-relational gradual pattern supp_τ is just a generalization of the case 2. Furthermore, the following theorem gives the *expected* supp_τ, denoted esupp_τ, in the case of statistically independent attributes. This expected support under independence can be used as a reference point to assess the quality of a mrGp.

Theorem 2. *(Expected* supp_τ*). Given a set of statistically independent numerical attributes* $s = \{A_1, A_2, \ldots, A_n\}$, $P_s = \{p_1, \ldots, p_m\}$ *is the set of all possible independent mrGps*[1] *generated from* s *where* $\circledast = \{>, <\} \cup \{=\}$. *The expected* supp_τ *of a pattern* $p \in P_s$ *is* $\mathrm{esupp}_\tau(p) = \frac{1}{(2^{n-1} + n(3^{n-1}-1)+1)}$.

Proof. After construction, we have that if $\circledast = \{>, <\}$ then for each element A_i there are two gradual items which are $A_i^>$ and $A_i^<$. Therefore, there are 2^n possible gradual patterns. However, for each gradual pattern p, there is always a complement pattern p' s.t. $\mathrm{supp}_\tau(p) = \mathrm{supp}_\tau(p')$. For instance, $A_x^>$ and $A_x^<$ or $A_i^> A_j^<$ and $A_i^< A_j^>$, etc, see Definition 39. Therefore, the number of independent patterns becomes $\frac{2^n}{2} = 2^{n-1}$. Additionally, $\forall i \in [1:n]$ s.t. $A_i^=$, there are 3^{n-1} possible gradual patterns formed from s without $\{A_i\}$ where $\circledast \in \{>, <\} \cup \{=\}$. Thus, there additionally are $n3^{n-1}$ independent gradual patterns. However, the pattern $\{A_1^=, \ldots, A_n^=\}$ is counted n times instead of once. Consequently, we have that the total number of independent gradual patterns is $(2^{n-1} + n3^{n-1} - n + 1) = (2^{n-1} + n(3^{n-1} - 1) + 1)$.

Let us assume that there is $(T^1 \bowtie \ldots \bowtie T^m)$ which is formed by joining all tables containing all attributes $A_1, \ldots, A_n$. Thus, $\forall p \in P_s : \mathrm{DomOcc}(p) = (\mathrm{Dom}(T^1 \bowtie \ldots \bowtie T^m), \mathrm{Dom}(T^1 \bowtie \ldots \bowtie T^m))$.

1. p and p' are independent mrGps if $\mathrm{DomOcc}(p) = \mathrm{DomOcc}(p') \wedge \mathrm{occ}(p) \cap \mathrm{occ}(p') = \emptyset$.

Table 7.1: Gradual support vs *Kendall's* τ support.

Hotel	price	distance from center
t_1	150	1
t_2	50	4
t_3	200	3
t_4	125	5

Given a pair of distinct tuples $(t_x, t_y) \in occ(p)$, $\nexists p' \neq p : (t_x, t_y) \in occ(p')$. Indeed, let us assume that $\exists A_i^{\circledast_1} \in p, A_i^{\circledast_2} \in p'$ s.t. $\circledast_1 \neq \circledast_2$, if $\pi_{A_i}(t_x) \circledast_1 \pi_{A_i}(t_y)$ then it is impossible to have $\pi_{A_i}(t_x) \circledast_2 \pi_{A_i}(t_y)$. So, $(t_x, t_y) \notin occ(p')$. Thus, we have that $\bigcap_{i=1}^{m} occ(p_i) = \emptyset$.

So, all the patterns $p \in P_s$ share the same domain and their occurrences are disjoint and therefore $\sum_{p \in P_s} supp_\tau(p) = 1$. Consequently, $\forall p \in P_s$ the $esupp_\tau(p) = \frac{1}{(2^{n-1}+n(3^{n-1}-1)+1)}$. □

7.4.2 Gradual Support

Until now, we have proposed the definition of *Kendall's* τ support for a multi-relational gradual pattern. However, in some case the *Kendall's* τ does not help us to fully understand the graduality of patterns.

For instance, see Table 7.1, let us assume the pattern $p = \{price^{<}, distance^{>}\}$ with the support threshold $\sigma_{min} = 0.6$. So we have the occurrence $occ(p) = 4$ and $supp_\tau(p) = \frac{2\times4}{4\times3} = 0.67 > \sigma_{min}$. Therefore, the pattern $p = \{price^{<}, distance^{>}\}$ is interesting and we can conclude that *"the more expensive hotels are, the nearer to the center the hotels are"* with 67%. However, *Kendall's* τ support does not emphasize the *consecutiveness* in changing of attributes over the values. Thus, we may only have granulative gradual tuple pairs, i.e. (t_1, t_2) is granulative since there is no other gradual tuple pairs such that (t_x, t_1) or (t_2, t_x), $t_x \in \{t_1, \ldots, t_4\}$. That is similar for $(t_1, t_4), (t_3, t_2)$ and (t_3, t_4). Therefore the pattern only rely on comparing two hotels and not for three or more than three hotels. However, in real world applications, we usually have a pool of objects to compare and in some specific situations $supp_\tau$ may not be effective.

Motivated by this issue, we propose a novel support for mrGps inspired by the previous work in [44] to evaluate the consecutiveness of graduality. First of all, let us define a list of tuples that support a gradual pattern as follows:

Definition 32. *(List of Ordered Tuples for a mono-rGp) [44]. Let* $q^i \in Q^i$ *is a mono-rGp. A list of tuples* $\mathcal{L} = \{t_1, \ldots, t_n\}$ *respects* q^i *if* $\forall i \in \{1, \ldots, n-1\}, (t_i, t_{i+1}) \in occ(q^i)$.

Definition 33. *(Gradual Support) [44]. Let* $\mathcal{G}_{q^i} = \{\mathcal{L}_1, \ldots, \mathcal{L}_h\}$ *be the set of all the lists re-*

specting a mono-rGp q^i. *The gradual support is defined as follows.*

$$supp_g(q^i) = \frac{max_{1 \leq e \leq h}(|\mathcal{L}_e|)}{|T^i|} \tag{7.12}$$

By generalizing the idea of Lisa et al., we propose the definition of list of ordered tuples for a mrGp as follows:

Definition 34. *(Correlating Lists of Ordered Tuples). Given two relations* T^i, T^j*, two lists of ordered tuples* $\mathcal{L}^{q^i} = \{t_1, \ldots, t_n\}, \mathcal{L}^{q^j} = \{t'_1, \ldots, t'_n\}$ *with* $q^i \in Q^i, q^j \in Q^j$. $\mathcal{L}^{q^i}$ *and* $\mathcal{L}^{q^j}$ *are called correlating each other if:*

a. 1:N relation. $PK^i \subseteq \mathcal{FK}^j$.

$$\forall x \in [1,n], \pi_{PK^i}(t_x) = \pi_{PK^i}(t'_x) \tag{7.13}$$

b. M:N relation. $\exists T^k \in db$ *s.t.* $(PK^i \subseteq PK^k$ *and* $PK^j \subseteq PK^k)$ *or* $(PK^i \subseteq \mathcal{FK}^k$ *and* $PK^j \subseteq \mathcal{FK}^k)$.

$$\forall x \in [1,n], \exists t_y \in T^k : \pi_{PK^i}(t_x) = \pi_{PK^i}(t_y) \wedge \pi_{PK^j}(t'_x) = \pi_{PK^j}(t_y) \tag{7.14}$$

Definition 35. *(Set of Lists of Ordered Tuples for a mrGp). Let* $p = q^1 \ldots q^m$ *is a mrGp. A set of lists of ordered tuples* $(\mathcal{L}^{q^1}, \ldots, \mathcal{L}^{q^m})$ *which respects* p *if* $\forall i \in [1,m] : \mathcal{L}^{q^i}$ *respects* q^i *and* $\forall j \in [1, m-1] : \mathcal{L}^{q^j}$ *correlates to* $\mathcal{L}^{q^{j+1}}$.

From Definitions 34 and 35, we can consider that $|\mathcal{L}^{q^1}| = \ldots = |\mathcal{L}^{q^m}|$.

Property 23. *(Anti-monotonicity) [44]. Let* q *and* q' *be two mono-rGps,* $\mathcal{G}_q$ *and* $\mathcal{G}_{q'}$ *respectively support* q *and* q'. *We have if* $q \subseteq q'$ *then* $max_{\mathcal{L}_e \in \mathcal{G}_q}(|\mathcal{L}_e|) \geq max_{\mathcal{L}_e \in \mathcal{G}_{q'}}(|\mathcal{L}_e|)$.

Note that the anti-monotonicity is also true for mrGps. Now, let us define the definition of mrGp gradual support as follows:

Definition 36. *(Gradual support of mrGp). Let* $db = \{T^1, \ldots, T^n\}$ *be a database for which each table* T^i *has a schema* $S(T^i) = (PK^i, \mathcal{FK}^i, \mathcal{A}^i)$.

1. Mono-rGp. $p = q^i$. *Let* $\mathcal{G}_p = \{\mathcal{L}_1, \ldots, \mathcal{L}_h\}$.

$$supp_g(p) = \frac{max_{1 \leq e \leq h}(|\mathcal{L}_e|)}{|T^i|} \tag{7.15}$$

2. Relational gradual pattern between two relations.

a. 1:N relation. $p = q^i q^j$ *where* $q^i \in Q^i$, $q^j \in Q^j$ *and* $PK^i \subseteq \mathcal{FK}^j$. *Let* $\mathcal{G}_p = \{(\mathcal{L}_1^{q^i}, \mathcal{L}_1^{q^j}), \ldots, (\mathcal{L}_h^{q^i}, \mathcal{L}_h^{q^j})\}$.

$$supp_g(p) = \frac{max_{1 \leq e \leq h}(|\mathcal{L}_e^{q^i}|)}{|distinct(PK^i, T^j)|} \tag{7.16}$$

b. M:N relation. $p = q^i q^j$ *where* $q^i \in Q^i$, $q^j \in Q^j$ *and* $\exists T^k \in db$ *s.t.* $(PK^i \subseteq PK^k$ *and* $PK^j \subseteq PK^k)$ *or* $(PK^i \subseteq \mathcal{FK}^k$ *and* $PK^j \subseteq \mathcal{FK}^k)$.

$$supp_g(p) = \frac{max_{1\leq e\leq h}(|\mathcal{L}_e^{q^i}|)}{min(|distinct(PK^i, T^k)|, |distinct(PK^j, T^k)|)} \tag{7.17}$$

c. Crossing element, Ω. $p = q^i\Omega$ *where* $q^i \in Q^i$, $\Omega \in Q^j$ *and* T^i, T^j *respects the cases 2a, 2b then we only need to transform the* $occ(p)$ *similar to Definition 31-2c*

3. mrGp. $p = q^1 q^2 \dots q^m$. *Let* $\mathcal{G}_p = \{(\mathcal{L}_1^{q^1}, \dots, \mathcal{L}_1^{q^m}), \dots, (\mathcal{L}_h^{q^1}, \dots, \mathcal{L}_h^{q^m})\}$.

$$\begin{cases} (1): \textit{If } q^1 q^2 \textit{ respects the case 2a then} \\ supp_g(p) = \frac{max_{1\leq e\leq h}(|\mathcal{L}_e^{q^1}|)}{|distinct(PK^1, T^2)|} \\ (2): \textit{If } q^1 q^2 \textit{ respects the case 2b then} \\ supp_g(p) = \frac{max_{1\leq e\leq h}(|\mathcal{L}_e^{q^1}|)}{min\left(|distinct(PK^1, T^k)|, |distinct(PK^2, T^k)|\right)} \end{cases} \tag{7.18}$$

Explanation and examples.

Property 24. ***(1:N relation).*** *Given a set of lists ordered tuples* $(\mathcal{L}^{q^i} = \{t_1, \dots, t_n\}, \mathcal{L}^{q^j} = \{t'_1, \dots, t'_n\})$ *respects to the pattern* $p = q^i q^j$*, we have that* $|\mathcal{L}^{q^j}| \leq |distinct(PK^i, T^j)|$.

Proof. Indeed, let us assume that $|\mathcal{L}^{q^j}| > |distinct(PK^i, T^j)|$, it means that $\exists t'_x, t'_y \in \mathcal{L}^{q^j}$ s.t. $\pi_{PK^i}(t'_x) = \pi_{PK^i}(t'_y)$ and thus $\exists t_x, t_y \in \mathcal{L}^{q^i}$ s.t. $\pi_{PK^i}(t_x) = \pi_{PK^i}(t'_x) \wedge \pi_{PK^i}(t_y) = \pi_{PK^i}(t'_y)$. Therefore, $\pi_{PK^i}(t_x) = \pi_{PK^i}(t_y)$ and so $(t_x, t_y), (t_y, t_x) \notin occ(q^i)$. Then, $\mathcal{L}^{q^i}$ does not respect q^i that violates the assumption. Consequently, we have that $|\mathcal{L}^{q^j}| \leq |distinct(PK^i, T^j)|$ and '=' happens when $\forall pk \in distinct(PK^i, T^j), \exists t_x \in \mathcal{L}^{q^i} : \pi_{PK^i}(t_x) = pk$. □

By applying Property 24, we have the |*all possible cases*| in this situation is $|distinct(PK^i, T^j)|$ and thus $supp_g(p) = \frac{max_{1\leq e\leq h}(|\mathcal{L}_e^{q^i}|)}{|distinct(PK^i, T^j)|}$.

For instance, see Figure 7.2, we have $\mathcal{L}_1^i = \{10, 11, 12\}$ the maximal tuple list respecting $q^i = \{Account.frequency^>\}$ and $\mathcal{L}_1^j = \{30, 32, 33\}, \mathcal{L}_2^j = \{30, 32, 34\}, \mathcal{L}_3^j = \{31, 32, 33\}, \mathcal{L}_4^j = \{31, 32, 34\}$ the maximal tuple lists respecting $q^j = \{Loan.amount^>\}$. Next, $(\mathcal{L}_1^i, \mathcal{L}_1^j)$, $(\mathcal{L}_1^i, \mathcal{L}_2^j)$, $(\mathcal{L}_1^i, \mathcal{L}_3^j)$, $(\mathcal{L}_1^i, \mathcal{L}_4^j)$ are maximal sets of lists of ordered tuples respecting to pattern $p = \{Account.frequency^>, Loan.amount^>\}$. Furthermore, $distinct(PK^i, T^{q^j}) = \{10, 11, 12\}$ and thus the support $supp_g(p) = \frac{3}{3} = 1$.

M:N relation. Essentially, M:N relationship is an adaptation of 1:N relationship so that we take the minimum number between $|distinct(PK^i, T^k)|$ and $|distinct(PK^j, T^k)|$.

Multi-relational gradual pattern. By applying Property 23, we have $q^1q^2 \subseteq p$ then $\max_{\mathcal{L}_e^{q^1} \in \mathcal{G}_p}(|\mathcal{L}_e^{q^1}|) \leq \max_{\mathcal{L}_e^{q^1} \in \mathcal{G}_{q^1q^2}}(|\mathcal{L}_e^{q^1}|) \leq |\mathtt{distinct}(PK^1, T^2)|$. Thus, $|$*all possible cases*$|$ in this situation is $|\mathtt{distinct}(PK^1, T^2)|$ and therefore $supp_g(p) = \frac{\max_{\mathcal{L}_e^{q^1} \in \mathcal{G}_p}(|\mathcal{L}_1^e|)}{|\mathtt{distinct}(PK^1, T^2)|}$. Similarly, we have the the $supp_g$ as in Equation 7.18.

Until now, we have proposed the multi-relational gradual pattern definition. Naturally, we can extract many patterns from a MRD $\mathtt{db}$. However, only some of them are interesting. In this chapter, we focus on extracting interesting patterns which satisfy the interestingness condition defined as follows:

Definition 37. *(Interesting Pattern). Given a predefined minimal support threshold* σ_{min} *and a pattern* $p = q^1q^2 \dots q^m$. p *is an interesting pattern if:*

$$\forall i \in \{1, \dots, m\} : supp(q^1 \dots q^i) \geq \sigma_{min} \tag{7.19}$$

where the notation $supp$ *is used to denote any of measures* $supp_\tau$ *and* $supp_g$.

For instance, given a pattern $p = q^1q^2q^3$, if p is an interesting pattern then $supp(q^1) \geq \sigma_{min}, supp(q^1q^2) \geq \sigma_{min}$ and $supp(q^1q^2q^3) \geq \sigma_{min}$.

7.5 Multi-Relational Gradual Pattern Mining Algorithms

Efficient extracting of complete set of mrGps is a non-trivial task. At first glance, the number of potential mrGps is exponential, i.e. approximately $2^{2\times|A_{db}|}$ where A_{db} is a set of all numerical attributes in the MRD $\mathtt{db}$.

Basic idea. Facing the huge potential search space, we propose an approach, named RGp. In RGp algorithm, we first extract mono-rGps from single tables. Then $\mathtt{db}$ is transformed into a graph $G = (V, E)$ where V is a set of vertices and E is a set of edges. Each vertex $v_i \in V$ stands for table $T^i \in \mathtt{db}$ and each edge can be considered as a relationship between two vertices (resp. tables) via keys and foreign keys. Next, we apply a depth-first search on the set of vertices V following E to combine mono-rGps together to be mrGps. In the process, we design three efficient rules which are *Complementary Pruning rule, Apriori Pruning rule* and *Backward Pruning rule* to avoid unnecessary computation and further search. Complementary Pruning is used to avoid the support computation for complement patterns (Definition 39). Then, Apriori Pruning is used to eliminate uninteresting patterns (Definition 37). Finally, Backward Pruning avoids computing redundant patterns which have been traversed before.

7.5.1 Mining Mono-Relational Gradual Patterns

To compute the mono-rGps in single relations, we apply *GRITE* algorithm [44] which is known as a very efficient algorithm. The main idea of *GRITE* is using a bitmap repre-

	t_1	t_2	t_3	t_4
t_1	0	0	1	0
t_2	1	0	1	1
t_3	0	0	0	0
t_4	1	0	1	0

(a) $M_{p_1=\{price^<\}}$

	t_1	t_2	t_3	t_4
t_1	0	1	1	1
t_2	0	0	0	1
t_3	0	1	0	1
t_4	0	0	0	0

(b) $M_{p_2=\{distance^>\}}$

	t_1	t_2	t_3	t_4
t_1	0	0	1	0
t_2	0	0	0	1
t_3	0	0	0	0
t_4	0	0	0	0

(c) $M_{p_3=\{price^<,distance^>\}}$

Figure 7.5: An illustrative example of Binary matrix of orders in Table 7.1.

sentation, named *binary matrix of orders*, to manage all gradual tuple pairs respecting a gradual pattern. The definition of binary matrix of orders is defined as follows:

Definition 38. *(Binary matrix of orders [44]). Let* p *be a mono-rGp,* $T_{\mathcal{G}_p}$ *is a set of distinct tuples in* $\mathcal{G}_p$. $\mathcal{G}_p$ *can be represented by a binary matrix* $M_{\mathcal{G}_p} = (m_{a,b})_{a\in T_{\mathcal{G}_p}, b\in T_{\mathcal{G}_p}}$, *where* $m_{a,b} \in \{0,1\}$.

If there exists an order relation between a and b, then the bit corresponding to the line of a and the column of b is set to 1, and to 0 otherwise. For instance, let us consider the gradual 1-attribute pattern $p_1 = \{Hotel.price^<\}$ in Table 7.1, $\mathcal{G}_{p_1} = \{\{t_2,t_4,t_1,t_3\},\{t_4,t_1,t_3\},\ldots,\{t_1,t_3\}\}$ and $T_{\mathcal{G}_{p_1}} = \{t_1,t_2,t_3,t_4\}$. This set of orders is modeled by means of a binary matrix of size 4×4, represented by Figure 7.5a. $M_{p_2=\{Hotel.distance^>\}}$ is illustrated in Figure 7.5b.

The joining between two mono-rGps p_1 and p_2 consists in computing all the common orders from two input matrices M_{p_1} and M_{p_2}. Common orders between matrices are represented by bits set to 1 for each of the input matrices. This can be achieved by the AND bitwise operation.

Theorem 3. *[44]. Let* $p = p_x p_y$ *be the mono-rGp generated using the two mono-rGps* p_x *and* p_y. *The following relation holds:* $M_p = M_{p_x}$ ***AND*** M_{p_y}.

Theorem 3 allows to efficiently perform the join operation using the level-wise method. Figure 7.5c shows the result of the join operation between M_{p_1} and M_{p_2}. This Theorem is also true for mrGps.

Essentially, the Kendall's τ-based pattern occurrence is the number of '1's in the binary matrix of orders that is easy to be computed. While the gradual occurrence, i.e. for $supp_g$, is the longest list from $M_{\mathcal{G}_p}$. To compute the gradual occurrence, GRITE proposes a *RecursiveCovering* function which is very efficient since each tuple from $M_{\mathcal{G}_p}$ is considered only once. Until now, we are able to efficiently evaluate the Kendall's τ-based pattern occurrence and gradual support-based pattern occurrence. Consequently, we can compute $supp_\tau(p)$ and $supp_g(p)$ by applying Definitions 31 and 36.

The search space of all mono-rGps can be eliminated by avoiding the computation of complement pattern support and uninteresting patterns, i.e. Properties 25, 26. Consequently, given a vertex v, Q^v is a set of interesting mono-rGps. Note that the crossing element Ω is included in Q^v.

7.5.2 Discovering Multi-Relational Gradual Patterns

In this section, we will clearly present RGp algorithm designed to extract mrGps. RGp is a depth-first search approach based on the set of vertices V following E to combine mono-rGps to be mrGps.

In Figure 7.7 the pre-order tree traversal is illustrated: tree nodes are labeled with numbers, denoting the depth-first search order. Each node is a list of vertices (resp. tables) $r = \{v_1, v_2, \ldots, v_n\}$. $route(r)$ is the route of r in the graph G, $route(r) = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n$ where v_1 and v_n respectively are the *head* and the *destination* of r, $head(r) = v_1$ and $dest(r) = v_n$.

In order to start the search process, vertices in V are collected into two different categories, source vertices and leaf vertices. The source vertices are not pointed from the other vertices while the leaf vertices do not point to any other nodes. The search process is started with source vertices. Note that the vertex v_{n+1} can be added into the node $r = \{v_1, \ldots, v_n\}$ if there is an edge from v_n to v_{n+1}.

Let us define some operations which will be used in the algorithm, note that '$\bowtie$' means Cartesian product, as follows:

– Q^r: given a current node $r = \{v_1, v_2, \ldots, v_n\}$, returns the set of all interesting mrGps at node r. Q^r is generated by combining mono-rGps in $Q^{v_1}, \ldots, Q^{v_n}$ such that $Q^r = Q^{v_1} \bowtie Q^{v_2} \bowtie \ldots \bowtie Q^{v_n}$.

– Given a superset r' of r, $r' = r \cup v'$ where $v' \notin r$, we have $Q^{r'} = Q^r \bowtie Q^{v'}$.

– $comm(r', r)$ and $diff(r', r)$: given two nodes r and r', $comm(r', r)$ returns the common relations between r' and r, i.e. $comm(r', r) = r' \cap r$. $diff(r', r)$ returns the relations that are in r' but not in r, i.e. $diff(r', r) = r' \setminus r$.

– $Q^r_{v_n}$: given a current node $r = \{v_1, v_2, \ldots, v_n\}$, return all interesting mrGps p at node r so that $\exists q \in p \wedge q \in Q^{v_n}$.

– $p \bullet p'$: given two mrGps $p = q^1, \ldots, q^m$ and $p' = q'^1, \ldots, q'^m$, return the extension of p by adding p' at the rear, i.e. $p \bullet p' = q^1, \ldots, q^m q'^1, \ldots, q'^m$.

Definition 39. *(Complement Pattern). Given a gradual variant $\circledast = \{<, >\}$ and its complement*[2] *$\overline{\circledast}$, two mono-rGps $q^i_x, q^i_y \in Q^i$ and two mrGps $p = q^1 q^2 \ldots q^m$, $p' = q'^1 q'^2 \ldots q'^m$. q^i_x and q^i_y complement each other, $q^i_y = \overline{q}^i_x$, if:*

$$|q^i_x| = |q^i_y| \text{ and } \forall A^{\circledast}_j \in q^i_x, \exists A^{\overline{\circledast}}_j \in q^i_2 \tag{7.20}$$

2. If $\circledast$ ='<' then $\overline{\circledast}$ ='>', $\circledast$ ='>' then $\overline{\circledast}$ ='<'.

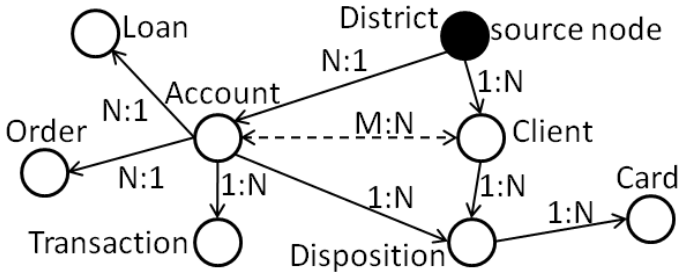


Figure 7.6: Graph of the Financial MRD in Figure 7.1.

p *and* p' *complement each other, denoted* $p = \overline{p'}$*, if:*

$$|p| = |p'| \text{ and } \forall q^i \in p, \exists \overline{q}^i \in p' \tag{7.21}$$

Property 25. *(**Complementary Pruning Rule**). Given two complement patterns* p *and* $\overline{p}$*, for* supp *any of the measures* $supp_\tau$ *and* $supp_g$*, we have* $supp(p) = supp(\overline{p})$.

Proof. After construction, we have $\forall (t_x, t_y) \in occ(p)$ then $(t_y, t_x) \in occ(\overline{p})$ and $\forall \mathcal{L}_e \in \mathcal{G}_p$ then $\overline{\mathcal{L}_e} \in \mathcal{G}_{\overline{p}}$ where $\overline{\mathcal{L}_e}$ is the reverse of $\mathcal{L}_e$. Thus, $|occ(p)| = |occ(\overline{p})|$ and $max_{\mathcal{L}_e \in \mathcal{G}_p}(|\mathcal{L}_e|) = max_{\mathcal{L}_e \in \mathcal{G}_{\overline{p}}}(|\mathcal{L}_e|)$. Additionally, $DomOcc(p) = DomOcc(\overline{p})$ and therefore the support formulas of p and $\overline{p}$ are the same. Consequently, $supp_\tau(p) = supp_\tau(\overline{p})$ and $supp_g(p) = supp_g(\overline{p})$. □

By applying Complementary Pruning, we can avoid computing the supports of complement patterns. Even though Complementary Pruning rule can eliminate the computation of a large number of candidates, there are many other candidates need to be pruned to shrink the search space. Essentially, if a mrGp p is not interesting then for any patterns $p \bullet p'$ is not interesting as well. Therefore, the extension process of p will be terminated. Consequently, we can eliminate a large number of pattern candidates. **Apriori Pruning rule** can be defined as follows:

Property 26. *(**Apriori Pruning Rule**). Given two mrGps* p *and* p'*, the following holds: If* p *is not interesting then* $p \bullet p'$ *is not interesting as well.*

Proof. After construction we have if p is not interesting then $\exists i \in \{1, \ldots, |p|\}$ s.t. $supp(q^1 \ldots q^i) < \sigma_{min}$. Additionally, $p \subseteq p \bullet p'$ and therefore $q^1 \ldots q^i \subseteq p \bullet p'$. Thus, $\exists i \in \{1, \ldots, |p \bullet p'|\}$ s.t. $supp(q^1 \ldots q^i) < \sigma_{min}$. So, $p \bullet p'$ is uninteresting. □

Now, let us define **Backward Pruning rule** which checks whether there are any previous routes that share the same destination with the current route, $route(r)$. If so, there are redundant patterns which can be ignored. For instance, at node labeled (10) we have

the current node $r = \{District, Account, Client, Disposition\}$. Essentially, all patterns generated from $Q^{District} \bowtie Q^{Account} \bowtie Q^{Disposition}$ have been evaluated before at the node labeled (7). Thus they are redundant and need to be pruned. The pruning rule can be defined as follows:

Property 27. ***(Backward Pruning Rule)*** *Given a current node* $r = \{v_1, \ldots, v_n\}$, $f = r \setminus \{v_n\}$ *is called father node of* r, *an already traversed node* r' *in the search tree so that* $dest(r') = dest(r) = v_n$ *and* $comm(f, r') \neq \emptyset$. Q^r *can be computed as follows:*

$$Q^r = (Q^{comm(f,r')} \bowtie (Q^{diff(f,r')} \setminus \Omega)) \bowtie Q^{v_n} \tag{7.22}$$

Proof. After construction, we have $Q^r = Q^f \bowtie Q^{v_n} = Q^{f \cap r'} \bowtie Q^{f \setminus r'} \bowtie Q^{v_n} = Q^{f \cap r'} \bowtie Q^{v_n} \bowtie ((Q^{f \setminus r'} \setminus \Omega) \cup \Omega) = (Q^{f \cap r'} \bowtie \Omega \bowtie Q^{v_n}) \cup (Q^{f \cap r'} \bowtie (Q^{f \setminus r'} \setminus \Omega) \bowtie Q^{v_n})$. Essentially, the patterns belong to $Q^{f \cap r'} \bowtie \Omega \bowtie Q^{v_n}$ have been traversed since r' has been traversed. Thus they can be discarded. Consequently, $Q^r = (Q^{comm(f,r')} \bowtie (Q^{diff(f,r')} \setminus \Omega)) \bowtie Q^{v_n}$. □

By applying Backward Pruning, we can discard numerous redundant patterns when there is another route passing the same destination with the current route. Furthermore, we can consider that $Q^{comm(f,r')} \bowtie (Q^{diff(f,r')} \setminus \Omega)$ is a set of patterns $p \in Q^f$ so that there exists a mono-rGp $q \subseteq p, q \in Q^{diff(f,r')}$ and $q \neq \Omega$. To select such patterns, we only need to scan all the patterns $p \in Q^f$ so that p must contain at least one gradual attribute in the tables $v \in diff(f, r')$. Note that we already have Q^f and therefore this pattern filtering step can be done efficiently.

For instance, see Figure 7.7, at node labeled 10, we have $Q^r = (Q^{\{District,Account\}} \bowtie (Q^{Client} \setminus \Omega)) \bowtie Q^{Disposition}$. Moreover, if we have a set of routes $R = \{r'_1, \ldots, r'_k\}$ so that $\forall r'_i \in R : dest(r'_i) = dest(r) \wedge comm(f, r'_i) \neq \emptyset$ we only need to apply Backward Pruning rule for each pair of routes r and r_i sequentially.

Backward Pruning is efficient in the sense that we only need to do filtering step once for r and not for its subtrees. This is because all the redundant patterns have been discarded from Q^f and thus Q^r-based Cartesian product can be applied for all the subtrees of r. To obtain the patterns, at each node $r = \{v_1, v_2, \ldots, v_n\}$, the set of interesting patterns $Q^r_{v_n} \subseteq Q^r$ will be reported whenever all the edges from v_n to other vertices have been traversed. For instance, see Figure 7.7, the patterns in $Q^{\{District,Account\}}_{Account}$ at node labeled (3) will be reported after step labeled (11).

Theorem 4. *(Identification of Relational Gradual Pattern in RGp). For a pattern* p, p *is an interesting relational gradual pattern if* p *passes Complementary Pruning, Apriori Pruning and Backward Pruning.*

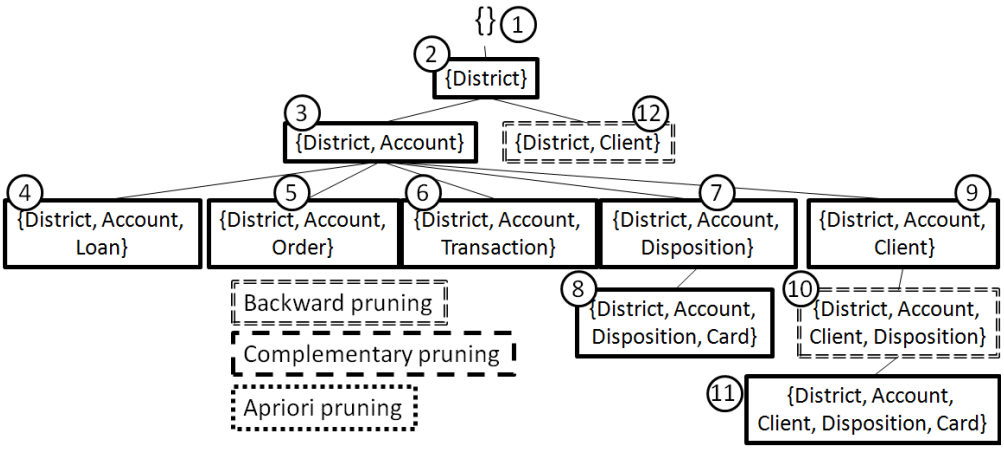


Figure 7.7: mrGp search space of the graph in Figure 7.6.

Proof. Clearly, every relational gradual pattern is derived by Complementary Pruning, Apriori Pruning, Backward Pruning. Now suppose that a pattern p passes all the conditions. First, Complementary Pruning ensures that p has not been evaluated yet. Also, we explicitly require p has not been traversed before by passing the Backward Pruning. Finally, p is an interesting pattern since it satisfies the Definition 37. Therefore p is an interesting relational gradual pattern. □

Theorem 4 makes the discovery of relational gradual patterns well embedded in the search process so that patterns can be reported on-the-fly without an extra post-processing step.

The RGp algorithm have two instances, i.e. τRGp and gRGp. The pseudo code of τRGp algorithm is presented in Algorithm 11. The main difference in gRGp algorithm is that we need to navigate a binary matrix of orders for each pattern to obtain the longest route. In the matrix navigation step, we also apply the navigation function presented in *GRITE*. The *Grad_combining* function of gRGp is described in Algorithm 12.

7.6 Experimental Results

A comprehensive experiment study has been conducted on real datasets which are the Financial data [3] (PKDD Cup 99) and the Thrombosis data[6] (PKDD Cup 2001). The Financial database scheme corresponds to the one given in Figure 7.1. There are 4,500 accounts, 5,369 clients, 5,369 objects in disposition, 6,471 objects in order, 2,500 objects in transaction, 682 objects in loan, 892 cards and 77 districts. The algorithms are implemented in Java, and the experiments are carried out on a 2.8GHz Intel Core i7 cpu, 4GB memory.

3. http://lisp.vse.cz/challenge/.

7.6.1 Multi-Relational Gradual Patterns

Several interesting patterns were discovered in the Financial database and Table 7.2 illustrates top strongest and longest patterns. We discuss some of them in the following.

Bases on $supp_\tau$: the τRGp algorithm returns the pattern $p = \{District.unemploy96^<, Order.Amount^>\}$ with $supp_\tau(p) = 58.32\%$ and the meaning is *"the less unemployed ratio districts are, the more expensive orders the habitants, who live in the districts, do"*. This pattern is quite interesting since it is useful to recommend for business men that the bigger deals are from the less unemployed ratio districts. Furthermore, by applying the Theorem 2, the $esupp_\tau(p) = 14.3\%$. Thus, the pattern has an attractive support value.

The other pattern related to the average salary is $p = \{District.avg\text{-}salary^<, Orders.amount^<\}$ with $supp_\tau(p) = 52.98\%$ and the meaning is *"the lower average salary the districts are, the less expensive orders the habitants, who live in the districts, do"*.

Bases on $supp_g$: the gRGp also returns many interesting mrGps. Furthermore, many of them are not in the top strongest patterns in terms of $supp_\tau$. For instance, the pattern $p = \{District.\#It\text{-}500^>, Loan.payment^<\}$ with $supp_g(p) = 44.444\%$ and the meaning is *"the more number of municipalities with inhabitants < 499 the districts have, the less the monthly payment is per inhabitant"*. This pattern is very interesting since it reveals unknown/unpublished loan policies.

The other interesting pattern is $p = \{District.\#It\text{-}500^>, Loan.duration^>\}$ with $supp_g(p) = 60\%$ and the meaning is *"the more number of municipalities with inhabitants < 499 the districts have, the longer debts the habitants have"*. This pattern shows that the habitants who are from the districts which have more number of municipalities with inhabitants < 499 are interested in borrowing money in longer time. Therefore, it could be useful to manage business strategies such as marketing, longterm investments, etc.

7.6.2 Efficiency and Pattern Distribution

To the best of our knowledge, there is no previous work which addresses mrGp. Therefore, in the efficiency experiments, we examine the proposed algorithms by varying the support threshold.

For Financial database, Figure 7.8a shows that τRGp algorithm is linear in terms of executing time. Furthermore, with the same support value, gRGp is more efficient than τRGp. The reason is that, by applying $supp_g$, there are less number of extracted patterns than $supp_\tau$, i.e. see Figure 7.8b. This is because $supp_g$ is tighter than $supp_\tau$ since the gradual tuple pairs are required to satisfy the consecutiveness constraint. However, there are differences in Thrombosis database. In most of cases, gRGp is slower than τRGp, i.e. see Figure 7.8c. Since the matrix size, i.e. $32,935 \times 32,935$, is significantly larger compare to the ones in Financial database, i.e. maximal matrix size is $6,471 \times 6,471$, the matrix navigation for computing the longest route is expensive. However, with low support values, i.e.

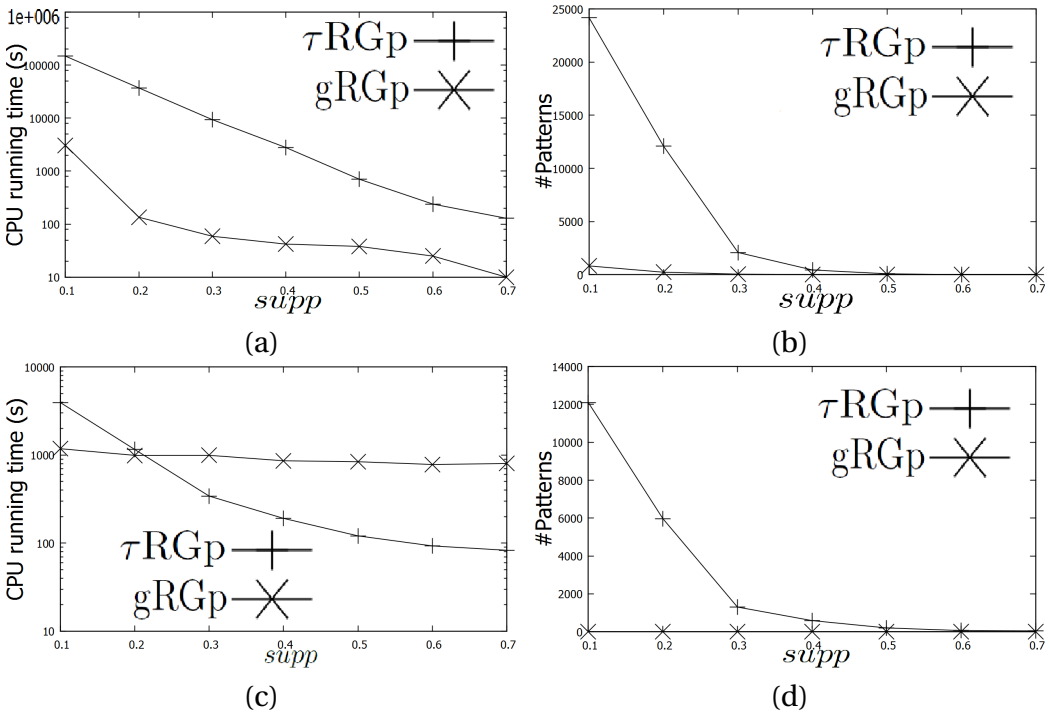


Figure 7.8: (a)(b)-Running time and #patterns on the Financial database. (c)(d)-Running time and #patterns on the Thrombosis database.

$supp = 0.1$, gRGp is faster than τRGp since there are many patterns extracted by applying $supp_\tau$, i.e. see Figure 7.8d.

Let us consider the effectiveness of Backward Pruning rule that is illustrated in Figure 7.9. We can consider that by proposing Backward Pruning rule, the large number of redundant patterns can be detected in Financial database, i.e. the number of redundant patterns is always larger than the number of extracted ones. One of the reasons is that we can avoid all the subtrees of $\{\texttt{District}, \texttt{Client}\}$, i.e. see Figure 7.7. Consequently, the process speeds up.

The distribution of mrGps extracted in Financial database is illustrated in Figure 7.10. We can consider that $supp_\tau$ and $supp_g$ are converged at low value area, i.e. $supp_\tau, supp_g \leq 0.3$. Furthermore, *the higher* $supp_g$, *the higher the* $supp_\tau$ *is.* In contrast, the higher value $supp_\tau$ does not mean that the pattern will obtain the higher $supp_g$. Consequently, if we only consider either $supp_\tau$ or $supp_g$ then we can lose many interesting patterns. Furthermore, both $supp_g$ and $supp_\tau$ are effective in mining useful mrGps.

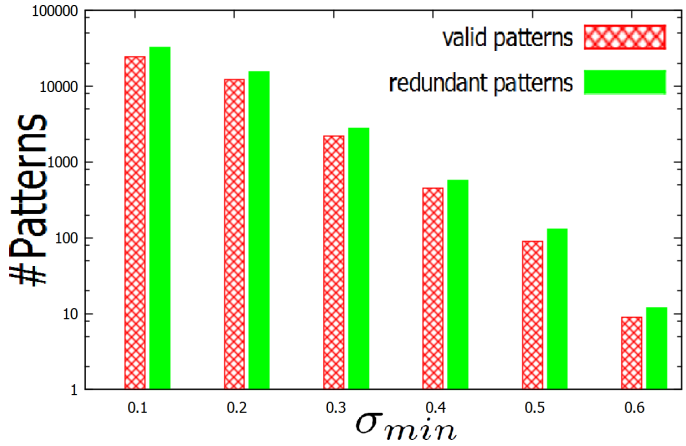


Figure 7.9: Number of redundant patterns in Financial database.

7.7 Related Work

Most previous work on relational pattern mining can be categorized into methods that generalize ideas from frequent itemset mining to the relational setting and methods that are based on Inductive Logic Programming (ILP). In this section we discuss the differences between these approaches and our approach as well as other works that do not fall into these two categories.

Well known ideas and algorithms from frequent itemset mining can be used for multi-relational data if applied on the join of all tables. This kind of patterns essentially is itemsets such that items are attribute values and transactions are the tuples of the join table [52] [48] [51]. A different approach is taken by Smurfig [54] where the support is measured with respect to every table, as the relative number of keys that the items correspond to.

Warmr [43] and Farmer [53] are ILP-based methods. The patterns are logic rules which can be regarded as local models of the database. The support is defined as the relative number of key values of one target table that satisfy the rule. The purpose of these methods is to extract the most frequent rules. This type of pattern is interesting and can illustrate well the relational structure.

R-KRIMP [51] patterns are similar to the work of [41], who define these patterns as simple conjunctive queries of the form: $[p_0,\ldots,p_n]$. RDB-Krimp [50] is an improvement of R-KRIMP. RDB-Krimp and RMiner [55] are methods for mining relational databases by using information theoretic ideas for the assessment of patterns. RDB-Krimp focus on the total description length of the database joint with the patterns, and patterns are deemed more interesting if they are better at compressing this description length. RMiner insteads deem patterns more interesting if they describe surprising aspects of the database in a concise way, which RMiner makes the results more relevant to an end-user.

Compare to the previous work, instead of working at attribute value level, we are inter-

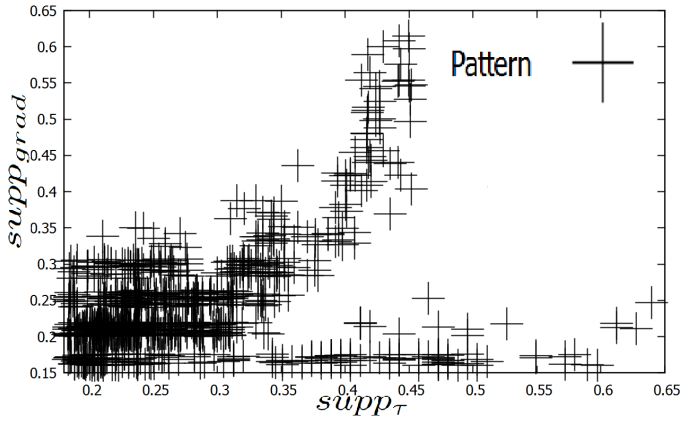


Figure 7.10: Multi-relational gradual pattern distribution with $\sigma_{min} \geq 0.15$.

ested in mining patterns at higher level. This kind of patterns is the correlation between attributes from a graduality point of view.

7.8 Discussion

In this chapter, we propose the relational gradual pattern concept which enable us to examine the correlations between attributes from a graduality point of view in MRDs. To efficiently mine interesting patterns, we also define its associated support measures based on $Kendall's\ \tau$ and gradual supports in MRD context. Then, τRGp and gRGp algorithms are proposed to completely cover and extract all the patterns. One of the future directions is to investigate top-k representative mrGp mining approach to avoid extracting huge amount of patterns. Another promising work is to define relational gradual pattern at tuple level or define contextual relational gradual pattern. That could help us better understand the structure of MRDs from a gradual point of view.

Algorithm 11: τRGp **algorithm**

```
   Input  : database db, source nodes S, G = {V, E}, σ_min
   Output : all mrGps
1  begin
2      R := ∅;
3      foreach v ∈ S do
4          r := {v};
5          RGpattern(db, r, v, G, σ_min);
6  RGpattern(db, r, v, G, σ_min)
7  begin
8      foreach node l ∈ v.edges do
9          combining(r, v, l);
10     output Q^v; delete Q^v; R := R ∪ r;
11 combining(route r, node v, node l)
12 begin
13     if type(v, l) = 1 : N then
14         denom := (|Join(PK^v, l)| choose 2);
15     else
16         denom := (|distinct(PK^v, k)| choose 2) × (|distinct(PK^l, k)| choose 2);
17     temp = ∅;
18     foreach pattern q^i ∈ Q^v do
19         if Backward(q^i, l, r, R) = true then
20             if exist(db, q^i) = false then
21                 create(db, q^i);
22             foreach pattern q^j ∈ Q^l do
23                 if complement(q^i q^j, temp) = false then
24                     create(db, q^j);
25                     occ := |q^i ⋈ q^j|;
26                     supp_τ(q^i q^j) := occ / denom;
27                     if supp_τ(q^i q^j) ≥ σ_min then
28                         temp := temp ∪ q^i q^j;
29                     delete q^j;
30                 else
31                     temp := temp ∪ q^i q^j;
32         delete q^i;
33     Q^l := Q^l ∪ temp;
34     RGpattern(db, r ∪ l, l, G, σ_min);
35 boolean Backward(q^i, l, r, R)
36 begin
37     foreach r' ∈ R do
38         if dest(r') = l then
39             flag := false;
40             foreach A* ∈ q^i do
41                 foreach p ∈ Q^{r\r'} do
42                     if A* ∈ p then
43                         flag := true; break;
44                 if flag = true then
45                     break;
46         if flag = false then
47             return false;
48     return true;
```

49 $type(v, l)$ is the connection type from v to l, $exist(db, q^i)$ returns true if q^i exists in database db, return false otherwise, $complement(q^i q^j, temp)$ returns false if there is no complement of $q^i q^j$ in temp.

Algorithm 12: Grad_combining

Input: route r, node v, node l

begin
  $temp = \emptyset$;
  foreach *pattern* $q^i \in Q^v$ **do**
    if $q^i.denom = null$ **then**
      if $type(v,l) = 1:N$ **then**
        $q^i.denom := |distinct(PK^v, PK^l)|$;
      else
        $q^i.denom := min(|distinct(PK^v, T^k)|, |distinct(PK^l, T^k)|)$;
    if $exist(db, q^i) = false$ **then**
      create(db, q^i);
    foreach *pattern* $q^j \in Q^l$ **do**
      if $Backward(q^i, l, r, R) = true$ **then**
        if $complement(q^i q^j, temp) = false$ **then**
          create(db, q^j);
          $update(q^i.matrix, q^i \bowtie q^j)$;
          $occ := longest(q^i.matrix)$;
          $supp_{grad}(q^i q^j) := \frac{occ}{q^i.denom}$;
          if $supp_{grad}(q^i q^j) \geq \sigma_{min}$ **then**
            $q^i q^j.denom := q^i.denom; q^i q^j.matrix := q^i.matrix$;
            $temp := temp \cup q^i q^j$;
          delete q^j;
        else
          $temp := temp \cup q^i q^j$;
    delete q^i;
  $Q^l := Q^l \cup temp$;
  $RGpattern(db, r \cup l, l, G, \sigma_{min})$;

Table 7.2: Interesting Patterns.

Kendall's τ : $supp_\tau$	(%)
$Account.date^{<}, Loan.date^{<}$	63.97
$District.unemploy96^{<}, Orders.amount^{>}$	58.32
$District.avg\text{-}salary^{<}, Loan.amount^{>}$	55.76
$District.\#crime95^{<}, Loan.amount^{>}$	54.42
$District.\#people^{>}, Loan.payment^{<}$	54.23
$Account.date^{<}, Loan.payment^{<}$	54.21
$District.\#crime96^{<}, Loan.amount^{>}$	53.97
$District.unemploy95^{<}, Loan.payment^{<}$	53.42
$District.avg\text{-}salary^{<}, Orders.amount^{<}$	52.98
$District.\#entrepreneur^{<}, Loan.payment^{>}$	52.31
Gradual : $supp_g$	(%)
$District.\#It\text{-}500^{>}, Loan.duration^{>}$	60
$District.\#It\text{-}500^{>}, Loan.date^{<}$	55.556
$District.\#people^{<}, Loan.duration^{>}$	50
$District.unemploy95^{>}, Loan.payment^{>}$	50
$District.\#crime95^{<}, Loan.duration^{>}$	50
$District.\#crime96^{<}, Loan.duration^{>}$	50
$District.unemploy96^{<}, Loan.date^{>}$	45.455
$District.\#It\text{-}500^{>}, Loan.payment^{<}$	44.444
$District.ratio\text{-}urban^{<}, Loan.payment^{>}$	44.444
Some of long patterns	$supp_\tau$(%)
$District.\{\#It\text{-}2000^{<}, \#crime95^{<}\}, Account.date^{>}, Loan.date^{<}$	36.91
$District.\{\#It\text{-}2000^{<}, \#crime95^{<}, \#crime96^{<}\}, Account.date^{>}, Loan.date^{<}$	36.53

CHAPTER

8 Applications

Preamble

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, these object sets are called spatio-temporal patterns. Analyzing such data has been applied in many real world applications, e.g., in ecological study, vehicle control, mobile communication management, etc. However, few tools are available for flexible and scalable analysis of massive scale moving objects. Additionally, there is no framework devoted to efficiently manage multiple kinds of patterns at the same time. Motivated by this issue, we propose a framework, named MULTI_MOVE*, which is designed to extract and manage different kinds of spatio-temporal patterns concurrently. A user-friendly interface is provided to facilitate interactive exploration of mining results. Since* MULTI_MOVE *is tested on many kinds of real data sets, it will benefit users to carry out versatile analysis on these kinds of data by exhibiting different kinds of patterns efficiently.*

Moreover, we also demonstrate that our proposed approaches and systems can be efficiently applied to many other domains such as gene expression analysis, and tweet user behavior analyzing. The GET_MOVE *approach was applied on 3 HIV time-series gene expression dataset to outline relationships between genes based on their expressions at different timestamps following infection. We have found that clustering gene expression data groups together efficiently genes of similar function based on their FC value and coherent results with our knowledge on HIV-1 versus HIV-2 infection were obtained. In additional, trajectories expressing the evolution of French political communities can be extracted from the political tweets which have been gathered during the French Election Campaign 2012.*

8.1 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 moving object data and has led to an increasingly large amount of data. Therefore, analysis on such data to find interesting patterns, called spatio-temporal patterns, is attracting increasing attention for applications such as movement pattern analysis, animal behavior study, route planning and vehicle control.

Despite the growing demands for diverse applications, there have been few scalable tools for mining massive and sophisticated moving object data. Even if some tools are available for extracting patterns (e.g. [33]), they mainly focus on specific kinds of patterns at a time. Obviously, when considering a dataset, it is quite difficult, for the decision maker, to know in advance which kinds of patterns are embedded in the data. To cope with this issue, we propose the MULTI_MOVE system to reveal, automatically and in a very efficient way, collective movement patterns like convoys, group patterns, closed swarms, moving clusters and also periodic patterns. Starting from the results of MULTI_MOVE, the user can then visualize, browse and compare the different extracted patterns through a user friendly interface.

8.2 The MULTI_MOVE System Architecture

The MULTI_MOVE general architecture, described in Figure 8.1, has three main layers: (i) collection and cleaning, (ii) mining, and (iii) visualization. The bottom layer is responsible for collecting and preprocessing moving objects. During this step, some cleaning and interpolations techniques are used to integrate and clean the raw data as well as to interpolate missing points. The interpolation techniques used are similar to the ones provided by most of spatio-temporal pattern mining algorithms.

MULTI_MOVE uses a new mining algorithm able to exploit the similar characteristics among different kinds of patterns. This new mining method combines both clustering and pattern mining to extract the final results. So, by applying it to the preprocessed data, MULTI_MOVE can automatically extract different kinds of patterns such as convoys, closed swarms, group patterns, moving clusters and periodic patterns.

The top layer is devoted to the visualization. MULTI_MOVE provides a platform for users to flexibly tune parameters and supports visualization of the results in different formats. The output can be written in Google Map [1] and Google Earth [2] formats to help users better explore the results. Furthermore, the system enables users to explore, plot and navigate over the different patterns in order to compare the differences among the various moving objects behaviors.

1. http://code.google.com/apis/maps/
2. http://earth.google.com/

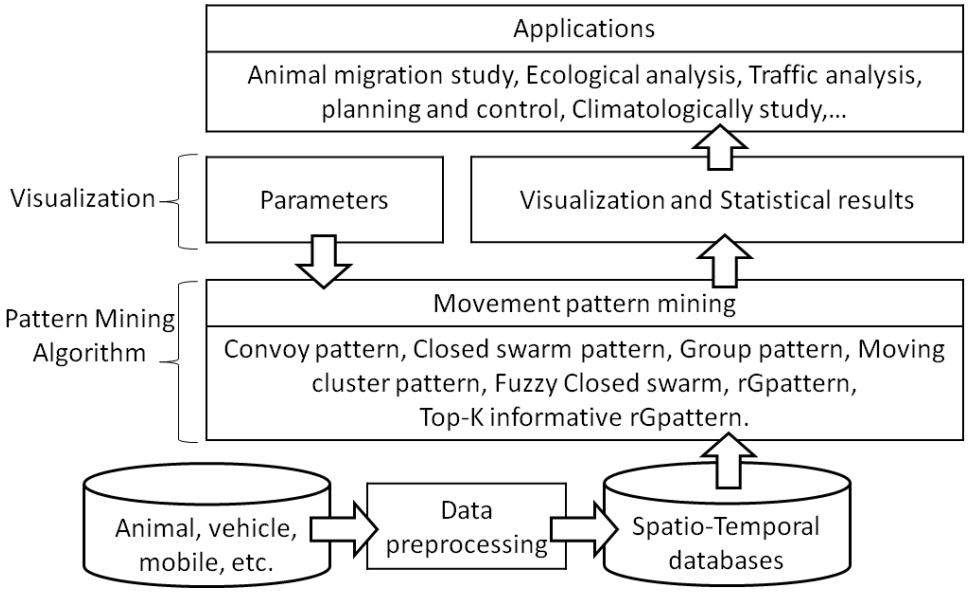


Figure 8.1: The MULTI_MOVE System Architecture.

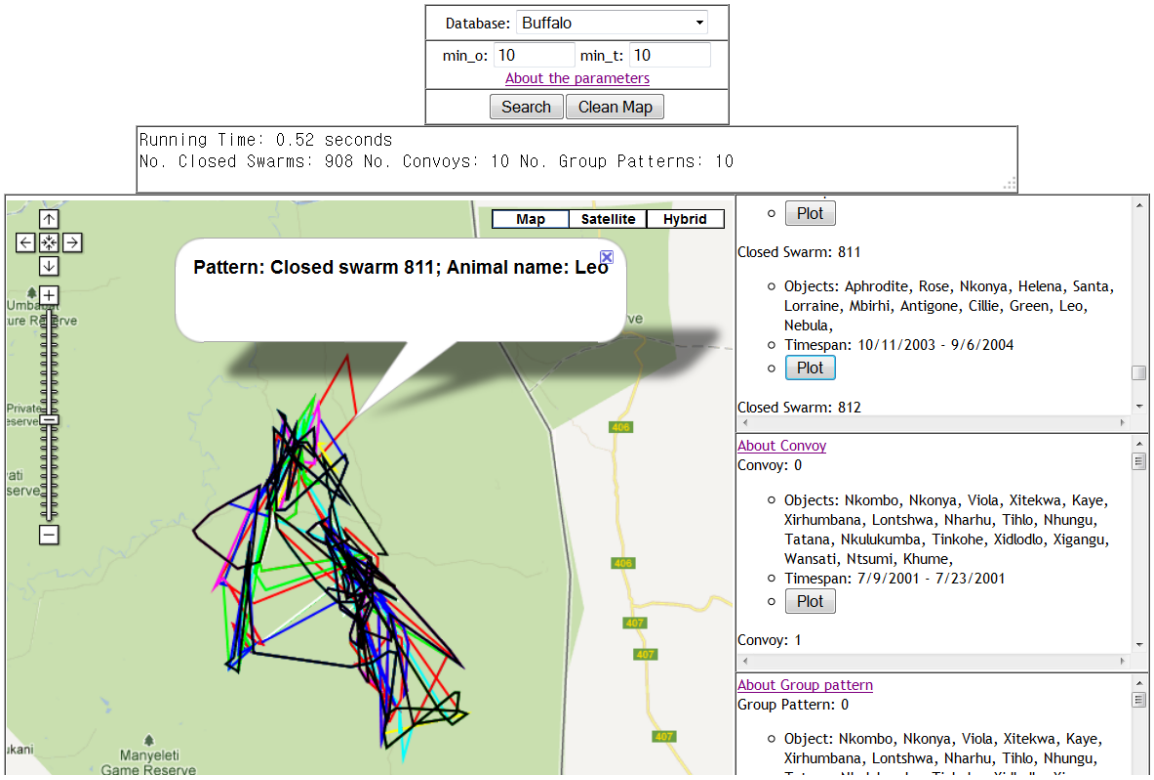


Figure 8.2: The Graphical Visualization Interface.

Figure 8.2 shows a screenshot of MULTI_MOVE [3]. It allows to choose a dataset and set the values of the parameter $\mathtt{min}_o$ and $\mathtt{min}_t$. In this example we have chosen the *Buffalo* dataset from the Movebank [4] project with $\mathtt{min}_o = 10$ and $\mathtt{min}_t = 10$ (combo box in the top of the figure). This dataset concerns the raw trajectories of 165 Buffaloes gathered from year 2000 to year 2006. In the second combox box in the figure, we can observe that, with these parameters, there are 908 closed swarms, 10 convoys and 10 group patterns. In this example, we observe that the execution time for extracting all these patterns is 0.52 seconds. From the combo box on right, the user can thus select a specific pattern and plot it on the main window. For instance, in our example, the user has decided to plot the 811^{th} Closed Swarm. Finally the user can have more information such as pattern identifier, animal name and timespan when clicking on plotted trajectories as illustrated in the main window.

8.3 Other Applications

8.3.1 Mining Trajectories on Genes

Detecting links between genes and diseases is a key challenge in human health in order to decipher disease process and identifying clinically pertinent biomarkers for its application in therapeutic purposes. The genome controls cellular processes such as genetic regulation, metabolic pathways, and signal transduction. Biological processes are very complicated involving numerous interaction between genes. Identifying those interactions in complex gene regulatory pathway is a challenging task. Thanks to DNA microarray technology, the study of gene expression has become considerably easier (Schena et al., 1995) enabling the simultaneous measure of the expression of thousands of genes through one experimentation. It offers an efficient tool to address complex disease since it provides an absolute overview of various cellular pathways activity simultaneously. Since its discovery, successful studies were carried out to identify genetic regulation and the different roles of particular genes (Chee et al. 1996, Chen et al. 1998, Duggan et al. 1999). Studying the unprecedented quantities of biological information produced by micro-array technology have contributed in elucidating and progressing complex biological research issues and providing new prospects for disease such as Cancer (Hoopes, 2008) or Alzheimer (Guttula et al., 2012). Most of studies focuses on identifying genes that showed a tendency to interact together and be functionally related (Barrenäs et al., 2012; Chen et al., 2008). Despite the efficiency of these methods, the most recurrent and challenging problems in mining the massive gene expression data are as follows:

1. Determining how the expression of any particular gene affects negatively or positively the expression of an other gene.

3. http://www.lirmm.fr/~phan/multimove.jsp
4. http://www.movebank.org/

2. Discovering novel genes implied in diseases.

3. Identifying changes in expression between different biological conditions. These challenges are even more important when analyzing time series (resp. temporal) experiments. Expressions are measured in a single sample at various points in time (Dequeant et al., 2008; Cooke et al., 2011). Temporal micro-arrays are particularly employed for analyzing time series data during a particular biological process, e.g. disease progression, for a better understanding of the impact of the genes on this process (Wang et al., 2008) and on the genetic alterations underpinning visual symptoms.

The AIDS (Acquired Immune Deficiency Syndrome) is caused by the human immunodeficiency virus, or HIV. Deaths from this disease increased throughout the years without any end in sight. Two majors type of HIV viruses exist: type-1 (HIV-1) and type-2 (HIV-2). HIV-2 is a close relative of the prototype AIDS virus HIV-1. HIV-2 is biologically similar to HIV-1, but precision concerning clinical outcomes of HIV-2 infected individuals are lacking. A prospective research, carried by Marlink et al. 1994, was conducted in women infected with HIV-2 and HIV-1 infection to determine and compare rates of disease development. Results showed that HIV-1-infected women had a 67% probability of AIDS-free survival in contrast with 100% for HIV-2-infected women 5 years after seroconversion. In addition, the rate of developing abnormal CD4+ lymphocyte with HIV-2 infection was also significantly reduced demonstrating that HIV-2 has a reduced virulence compared to HIV-1 (Marlink et al. 1994). HIV-1 cells invasion is enabled by the binding of envelope glycoproteins to the receptor CD4 and a co-receptor, principally CXCR4 or CCR5, according to the viral strain (X4 or R5, respectively). Invariably, differences are observed between HIV-1 and HIV-2 but their molecular bases are still largely unknown. However, the characterization of these differences and involved cellular regulation could lead not only to better understand the pathophysiology of the virus but also to develop new anti-HIV strategies. It's possible that mRNA modulations induced by X4, R5 and HIV-2 may be different and participate in replication differences between theme. In this context, it's crucial to develop methods to asses the understanding of the modulation of the transcription following infection with the three strains of HIV virus.

Despite the availability of hundreds of algorithms for analyzing such data, it still remains a challenging task notably because of the lack of a consensus about the most suitable strategy for clustering temporal micro-array. The objective of this section is to develop an original process to extract, annotate and visualize relevant molecular bio-markers to distinguish both HIV types. Molecular bio-markers are generated from a time series micro-arrays and are based on our trajectory pattern mining approaches.

Indeed, the INCREMENTAL GET_MOVE is adopted to identify genes with *"similar"* expression level at different time points. In essential, we detect moving clusters of genes. Considering that co-expressed genes are very frequently co-regulated as well, and the identification of co-expressed genes leads to discover and understand the regulatory mechanisms of the studied disease.

Table 8.1: Example of matrix query to compare extracted trajectories for HIV-2, R5, and X4.

	Dataset
HIV-2	∅
R5	X
X4	X

Table 8.2: Cluster matrix corresponding to gene dataset.

T_{db}		04h	08h		24h	48h	72h
Clusters C_{db}		c_1	c_2	c_3	c_4	c_5	c_6
Genes	$gene_1$		1			1	1
	$gene_2$		1		1	1	1
	$gene_3$	1		1	1	1	1
	$gene_4$	1		1	1	1	

To investigate the suitability of theses approaches in the analysis temporal micro-array data, we utilized the expression profile of about 19.000 human genes at 5 points timestamps after infection by one of 3 strains of HIV (HIV-2, X4, R5). To study the general and specific transcriptomic modulation between all the strains of HIV virus, we enable the biologist to query the closed swarms. Table 8.1 illustrates the example of such query where the "X" denotes the query will retrieve the patterns in the train, while "∅" and "-" respectively denote "*non inclusion*" and "*non interest*".

In order to extract closed swarms, genes will be presented in a cluster matrix illustrated in Table 8.2; then GET_MOVE can be applied directly. Although the semantic of extracted patterns is relevant for the biologist, the huge amounts of extracted patterns are not easy to process and to interpret. Therefore, we propose new visualization tools [5] (Figure 8.3) to facilitate the analysis of extracted patterns by the biologist. For that, we annotated the different type of patterns following their biological process that we displayed graphically allowing their visualization in an intuitive form for the biologist. Also, this visualization allowed the visualization of both common and specific trajectories between HIV strains (HIV-2, HIV-1, X4 and R5).

Figure 8.3 shows the screenshot of the tool. It allows to choose patterns to visualize and also enable us to select the patterns correlated to specific genes in which we are interested. Also, user can upload and visualize their own data and many other functions.

8.3.2 Mining Trajectories on Tweets

Usually pattern mining approaches focus on extracting different kinds of patterns, i.e. itemsets, sequences, trees, graphs, etc., hidden in the database. Using these patterns for

5. http://www.lirmm.fr/~phan/interface/interface/interface.html

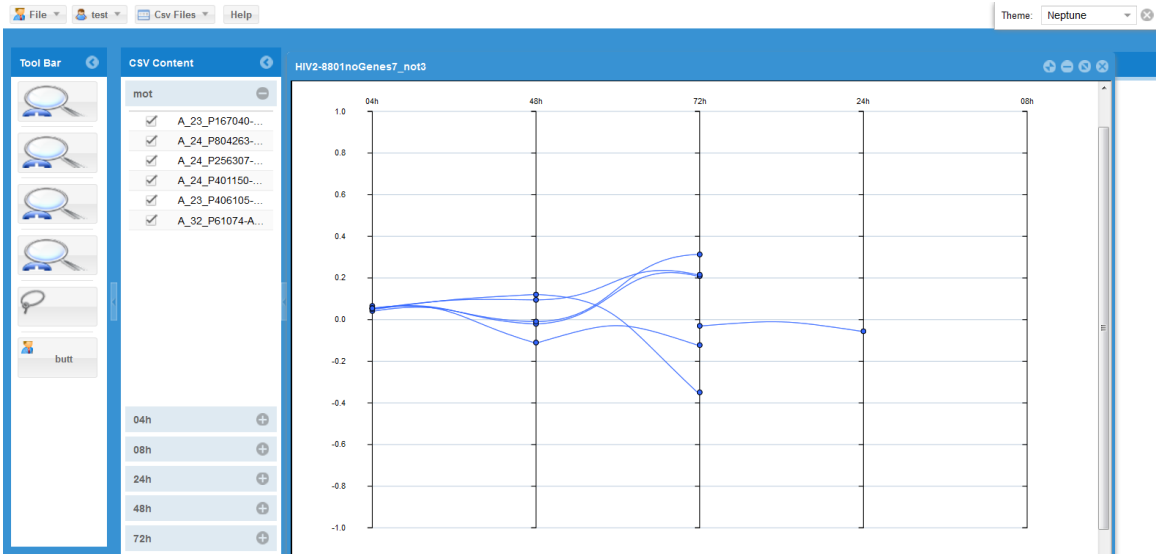


Figure 8.3: The graphical visualization interface of gene expression tool.

analyzing tweets is one of the topic addressed by some research work. In this study, our main objective is to highlight that spatio-temporal patterns extracted from tweets can be very useful for the decision maker.

Recently, we defined a new project called PoloP [6] (Political Opinion Mining) which aims to cope with the analysis of the evolution of French political communities over Twitter during 2012 both in terms of relevant terms, opinions, behaviors. 2012 is particularly important for French political communities dues the two main elections: Presidential and Legislative. From the 12th December 2011 to the 19th June 2012, we thus obtained 2,122,012 tweets from 213,005 users. For 130,618 tweets, 232 users can unambiguously be assigned to a political party (i.e. user is a politician or an official political community account). By using our defined measures [66], we can select for different political parties the set of relevant words at different periods (see Figure 6). In order to extract interesting trajectories for different parties we applied a clustering technique, for each party, on the top-k set of relevant terms over time. Thus, we group together users from the same party sharing the same words.

Figure 7 illustrates a kind of trajectory that might be extracted from the political tweets. In February 2012, in one of the first debate, the focus of the French election campaign suddenly shifts from the economy to racism and national identity. For the right party, as we can notice, during the debate, most of used words are such that lots of party members shared the same words. Just after the debate, some right party users move to another cluster where words such as *"national identity"*, *"France"* while other ones shift to other topic

6. http://www.lirmm.fr/~bouillot/polop/polop.html

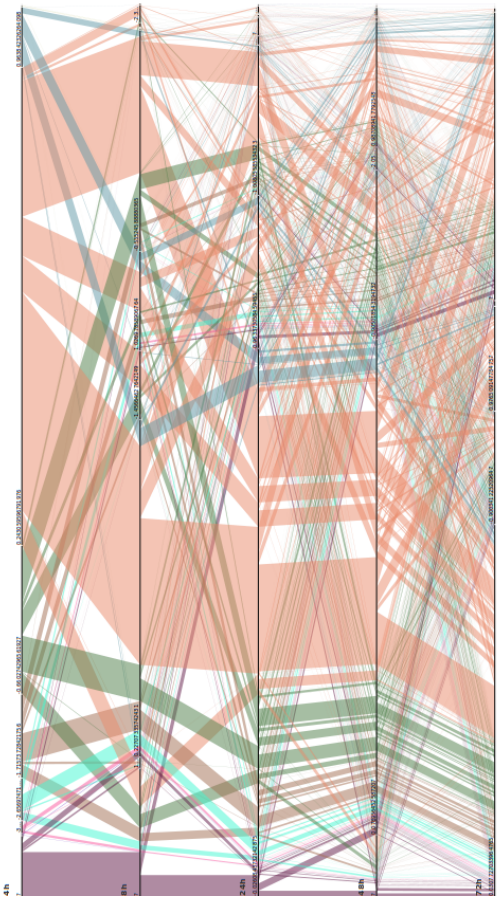


Figure 8.4: Another graphical visualization interface of gene expression tool.

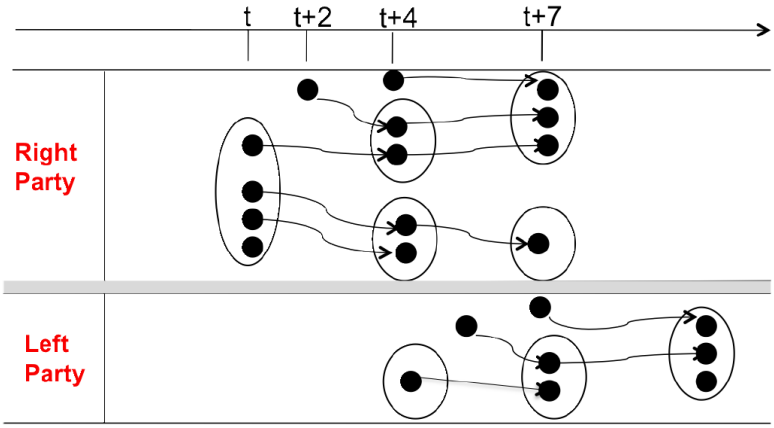


Figure 8.5: An example of a trajectory that can be extracted from tweets.

like *"elections"*. Actually, during the debate the main trend was to motivate some electors from the extreme right. During this time, as we can notice, in the left party, as it is illustrated through the gradual pattern, the left party uses completely different kinds of words but just when right users behavior split.

8.4 Discussion

In this chapter, we propose a system, MULTI_MOVE, which is designed to automatically and efficiently extract different kinds of spatio-temporal patterns at the same time. Starting from these results, the user can browse, navigate and compare different patterns in an easy way. The comparative analysis allows the user to understand and discover which results can fit better her researches. All this part has been defined in order to highlight the results of the previous chapter. Interestingly, we also show that our proposed systems and approaches can be applied on diverse applications such as bioinfomatics and online social network analysis. For instance in biology we show that thank to trajectories new unknown knowledge can be extracted. Even if in this work the idea was to first apply our approach in a very different context, the most interesting part is that biologist have found that very interesting new correlations have been extracted. Some of them were already known by biologist. This is typically a first way to validate the results we obtain. Some others are news. Biologists were very interested by these new correlations. Now they are investigating through biological experiments the relationships with genes that we have highlighted.

Experiments that have been conducted on tweets has been defined to evaluate if trajectories can be useful for the decision maker. Even if all the experiments are not reported on this report, it is obvious that trajectories are able to extract real behaviors over time. Recently new approaches have been defined for analyzing the behavior of user or communities over time. Here also with trajectories we shown that we are able to extract well-known behaviors. The most interesting thing is that automatically we are able to extract unexpected or surprising patterns. In this part our goal was not to fully investigate the analyzing of tweets over time but rather to illustrate and show that trajectories are a new way to better understand the behaviors. In the conclusion of this report we propose other applications that can take advantage of trajectories.

CHAPTER

9 Conclusion & Perspectives

Preamble

Until now, I have addressed several key research questions however there are many challenging issues I need to deal with in my long term research career. In this chapter, I will summary our work and clearly present my future perspectives.

9.1 Conclusion

In this work, we focus on mining and managing movement patterns from moving object data. The three steps framework has been proposed, i.e. see Figure 1.2. In the first step, we aim to mine and manage different traditional movement patterns such as convoys, closed swarms, group patterns, moving clusters, and periodic patterns. A unifying incremental algorithm, named GeT_Move, is presented that enable us to extract these patterns concurrently. To improve the efficiency, we also proposed *Parameter Free* GeT_Move and *Object Movement Pattern Mining Algorithm Based on Explicit Combination of FCI Pairs*.

In the second step, we aim to access the relevant of movement patterns. Since the existing pattern models either require consecutive time constraint or completely relax this constrain that results in loosing meaningful patterns or extract many extraneous ones. To cope this issue, we propose the notion of *fuzzy moving object clusters* in order to deal with time gaps. The obtained patterns are in the form of *"The group of objects are moving together from A to B with 60% weak, 30% medium and 10%strong time gaps"*. Then an novel property which can be integrated into GeT_Move to extract the complete set of fuzzy moving object clusters.

Another meaningful work in this step is that we present the novel movement pattern concept, named *gradual trajectory pattern*, to capture the object moving trends. The dis-

covery of all the maximal gradual trajectory patterns is a non-trivial task due to very huge search space, i.e. $2^{|C_{db}|}$. To cope this issue, we propose ClusterGrowth algorithm with three efficient pruning rules, i.e. *Apriori Pruning, Backward Pruning* and *Actual Maximal Checking*, to optimize the search space. In order to enrich the utility the gradual trajectory pattern concept, we further propose an MDL principal-based algorithm, named DiCompoGp, to directly extract top-K informative gradual trajectory patterns. An novel encoding scheme which allows overlapping between rGpatterns has been proposed and that improved the compression quality of the proposed approaches.

After managing all the traditional movement patterns, fuzzy moving object clusters and gradual trajectory patterns, there are still a huge amount of patterns that poses limit in there usefulness. To address this issue, in the third step of our framework, we propose an novel approach, named SmartCompo, to mining compressing movement patterns. By considering different kinds of overlapping patterns at the same time, we have more insightful structure of the moving behavior of objects.

The potential utility of the developed spatio-temporal mining tools must be justified in their applicable field. Therefore, we develop the MULTI_MOVE system to reveal, automatically and in a very efficient way, collective movement patterns like convoys, group patterns, closed swarms, moving clusters and also periodic patterns. Starting from the results of MULTI_MOVE, the user can then visualize, browse and compare the different extracted patterns through a user friendly interface in Google Maps and Google Earth. The MULTI_MOVE system has gained a lot of publicity, since it is the first system that enable users to obtain the comparative results.

As an extra work, we further propose the relational gradual pattern concept which enable us to examine the correlations between attributes from a graduality point of view in MRDs. To efficiently mine interesting patterns, we also define its associated support measures based on Kendall's τ and gradual supports in MRD context. Then, τRGp and gRGp algorithms are proposed to completely cover and extract all the patterns.

9.2 Streaming GeT_Move: Mining Representative Movement Patterns from Streaming Trajectory Data

In GeT_Move, we assume that the object set O_{db} is remained the same during the time. However, in real world context such as cars in highway, some cars could be leave while the other cars joint the highway. Thus, the set of objects O_{db} is flexible by the time. Another case study is that new object data can be available after the generating of closed itemsets and patterns and therefore O_{db} and the cluster matrix will be changed. The challenging issue here is that the support value of the pattern could be changed and even more serious if the the set of clusters C_{db} is also changeable. What we can show here is that if O_{db} is flexible then C_{db} and of course the final results will be changed by the time. Thus,

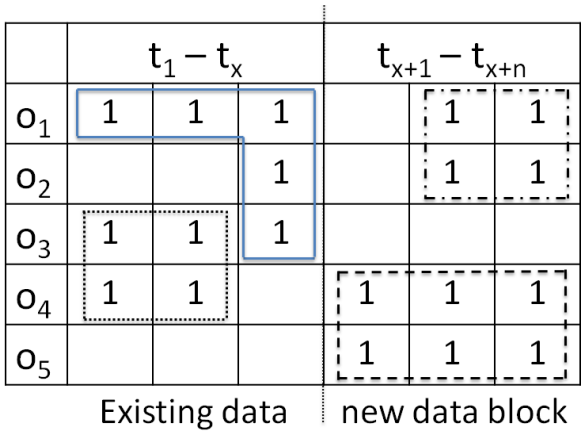


Figure 9.1: An example of Streaming GeT_Move.

we need to re-execute all the system whenever there are any differences in O_{db} that is cost prohibitive and time consuming.

To tackle the problem, we can apply data stream technique such that we summarize the existing data, which is considered as a block, with representative movement patterns, denoted $\mathcal{P}$. If there are new data block available, it will be represented by a set of compressing movement patterns $\mathcal{P}'$ which is used to combine with existing movement patterns $\mathcal{P}$ to be a global data. From the global data, we can directly extract representative patterns. Our example is illustrated in Figure 9.1. Refer to Section 9.4 for the *directly mining representative movement patterns* algorithm.

By doing so, we can keep the representative movement patterns for different historical data in terms of time. For instance, $T_{db} = \{T_1, T_2, \ldots, T_3\}$, we are able to retrieve the informative behavior of moving objects in $[T_1, T_2], [T_1, T_2, T_3]$, etc. That information will be very meaningful for analytics. Furthermore, to improve the efficient of the approach, objects O_{db} can be indexed by using *kd-tree* and when new objects are available, they will be added into the *kd-tree* as new nodes.

For instance, see Figure 9.1, there are two representative patterns from the existing data and the other two patterns from new data block. All of them together will be the general data from which the final compressing movement patterns can be extracted. This future work is a result from a brainstorming meeting between our team and Dr. Albert Bifet, Yahoo! Research Lab-Barcelona, Spain.

9.3 CorGpattern: Combined Time Relaxed Gpattern

Naturally, an rGpattern can follow by the other rGpatterns and combining them might propose interesting insight. For instance, with reference to Figure 9.2, there are two po-

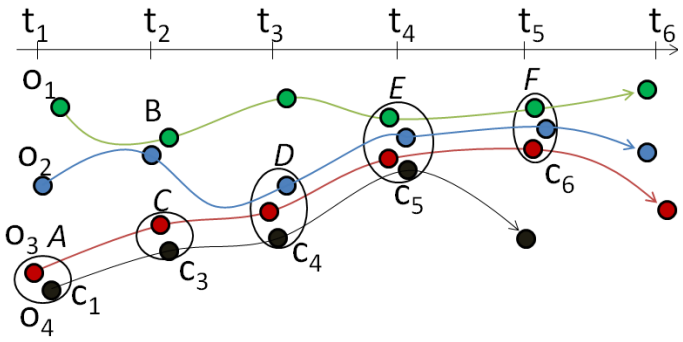


Figure 9.2: An example of CorGpattern extracted from our running example, i.e. Figure 1.1.

tential rGpatterns which are $C_1 = \{c_1, c_3, c_4, c_5\}$ and $C_2 = \{c_5, c_6\}$. We can consider that C_1 is followed by C_2 and if we combine the two patterns we can obtain a new pattern such as: *"from* t_1 *to* t_5*, as time passes, the more objects are following the trajectory* $\{A \rangle C \rangle D \rangle E\}$ *then the less objects are following the trajectory* $\{E \rangle F\}$*"*. In this pattern, the location E is the most important place for all the objects since they get together there as a meeting point and leave the group after a while. To figure out such patterns and meaningful locations, we further propose the notion of CorGpattern, i.e. *combined time relaxed gradual moving object clusters*, which essentially is a list of rGpatterns correlating each other in terms of time periods and objects.

Until now, we have defined two kinds of rGpatterns, i.e. rGpattern$^{\geq}$ and rGpattern$^{\leq}$, which can capture the object moving trends. Moreover, combining rGpatterns might reveal unknown object movement behavior as well as interesting locations. Indeed, in different applications CorGpatterns and these locations have different meaning. For instance, in animal migration these locations could be water sources or nests for birds, in indoor movement analysis they could be meeting room or personal office, in traffic data they could be events, car parking or customer home for taxis, etc.

Similar to rGpattern, we only focus on extracting the complete set of maximal CorGpatterns to avoid redundancy. To extract all the maximal CorGpatterns, we propose an algorithm, named CoClusterGrowth, which is an modification of ClusterGrowth. In CoClusterGrowth, *Graduality Pruning* and *Backward Pruning* are modified to be suitable to the characteristics of CorGpattern.

9.3.1 CorGpattern Definition

Essentially, a CorGpattern is a list of alternating maximal rGpattern$^{\geq}$s and rGpattern$^{\leq}$s which are coherent together in terms of time and objects. Formally, the CorGpattern can be defined as follows:

Definition 40. ***(CorGpattern)***. *Given a list of rGpatterns* $\mathcal{G} = \{C_1^{*_1}, \ldots, C_n^{*_n}\}$*, a sliding window* w*.* $\mathcal{G}$ *is called a CorGpattern if:*

$$\begin{cases} \forall i \in \{1, \ldots, n-1\}, \\ (1): *_i = \bar{*}_{i+1}. \\ (2): \begin{cases} \textit{if } *_i = \geq: o(tail(C_i^{*_i})) \supseteq o(head(C_{i+1}^{*_{i+1}})). \\ \textit{if } *_i = \leq: o(tail(C_i^{*_i})) \subseteq o(head(C_{i+1}^{*_{i+1}})). \end{cases} \\ (3): t(head(C_{i+1}^{*_{i+1}})) - t(tail(C_i^{*_i})) \leq w. \end{cases} \tag{9.1}$$

where `tail` *and* `head` *functions respectively return the last and the first cluster in a rGpattern.*

For instance, in Figure 9.2, we have several potential CorGpatterns as $\mathcal{G}_1 = \left\{C_1^{\leq} = \{c_1, c_3, c_4\}, C_2^{\geq} = \{c_5, c_6\}\right\}$, $\mathcal{G}_2 = \left\{C_1'^{\leq} = \{c_1, c_3, c_5\}, C_2^{\geq} = \{c_5, c_6\}\right\}$, etc. Moreover, $\mathcal{G}_1$ can also be presented as a list of clusters, denoted $c\mathcal{G}_1 = \{c_1, c_3, c_4, c_5, c_6\}$. $c\mathcal{G}$ essentially is a distinct list of clusters belonging to all C^* in $\mathcal{G}$.

In this section, we concern about mining the maximal CorGpatterns in terms of distinct list of clusters. The basic idea is that if $\mathcal{G}$ is a CorGpattern, it is unnecessary to output any subset $\mathcal{G}'$ of $\mathcal{G}$ even if $\mathcal{G}'$ may also satisfy CorGpattern requirements.

Definition 41. ***(Maximal CorGpattern)***. *Given a CorGpatterns* $\mathcal{G}$ *with* $c\mathcal{G} = \{c_1, \ldots, c_n\}$*.* $\mathcal{G}$ *is maximal if* $\nexists \mathcal{G}' : c\mathcal{G} \subset c\mathcal{G}'$ *and* $\mathcal{G}'$ *is a CorGpattern.*

9.3.2 CoClusterGrowth: Discovering Maximal CorGpatterns

Basic idea of CoClusterGrowth. Continue with the valid leaf node by changing the variation $*$, report if the node cannot be enlarged, i.e. Apriori Pruning. If exists a cluster c so that $t(c) < t(c_n)$ and exists $c\mathcal{G}' = c\mathcal{G} \cup c$ and $\mathcal{G}'$ is a CorGpattern, then any subtrees of the current node are pruned, i.e. Backward Pruning. If exist c' so that $0 < t(c') - t(c_n) \leq w$ and $c\mathcal{G}' = c\mathcal{G} \cup c$ and $\mathcal{G}'$ is a CorGpattern then $\mathcal{G}$ is not maximal, i.e. Actual Maximum Checking. If all the rules are passed then it is a maximal CorGpattern, i.e. Theorem.

9.4 Directly Mining Representative Movement Patterns through Compression

Even if SmartCompo is an effective algorithm, it is a straightforward approach and we need to handle the efficiency issue. Indeed, with dense and large datasets, the number of generated patterns is exponential and thus efficiency issue will be a key challenge.

To address the issue, we can adapt DiCompoGp with multi-pattern structure encoding scheme. In DiCompoGp, a list of cluster $\mathcal{C} = \{c_1, \ldots, c_n\}$ which can be a rGpattern$^{\leq}$, rGpattern$^{\geq}$ or a closed swarm which have the same encoding scheme presented in

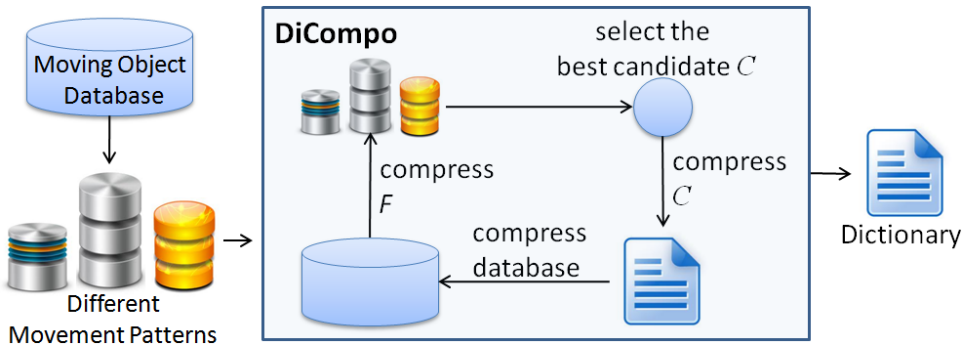


Figure 9.3: DiCompo algorithm in action.

SmartCompo, i.e. Property 21, however for overlapping allowing encoding scheme is a little bit different. See Figure 9.3, we can consider that after selecting the best candidate $\mathcal{C}$ by employing DiCompoGp, $\mathcal{C}$ will be compressed given the dictionary $D = \{p_1, p_2, \ldots, p_n\}$ from the bottom p_1 to the end p_n consequently. By doing this, even $\mathcal{C}$ overlaps with the other patterns in D, it can be used to compress the data actually. If the encoded $\mathcal{C}$ improve the data compression then it will be added into the dictionary D, and rejected otherwise.

Even the idea has been figured out, we need to finalize all the definitions, properties and theorems. Also, the algorithm must be implemented and tested on real world datasets to access the effectiveness and efficiency.

9.5 Completed Mining Multi-Relational Gradual Patterns

Until now, we have presented an novel concept of multi-relational gradual patten and an effective algorithm RGp to extract the pattern directly from database db. However, meaningful patterns are still loosing. Indeed, recently we only can extract the patterns following the existing relation scheme in relational database theory, i.e. 1:N and M:N which essentially is 1:M & N:1. However, in real context there is another M:N relation which is composed by M:1 & 1:N.

For instance, see Financial database in Figure 7.1, we can consider that the relationship between Loan and Order essentially is M:N and they share the same accountid field. They essentially are characters of Account table. One of the potential pattern is $\{\text{Loan.payment}^{>}, \text{Order.amount}^{>}\}$ and the meaning is *"the more loan payment the accounts have, the more expensive orders the account owners do"*. Such kinds of patterns are very meaningful.

The discovery of these patterns is not an trivial task since it demands an novel support definition and propagation approach. Furthermore, the search space will be very large as well. For instance, see Figure 7.6, we can have some pattern from Loan and Card relations. Thus, what we can show here is that the search space will exactly be $2^{|db|}$ and the

number of pattern candidates is $2^{2\sum_{i=1}^{n}|A^i|}$ *without running rules* since the existing ones are ineffective in this case.

To address these challenges, we need an innovative way to estimate the support of the patterns to prevent unnecessary computation, an novel propagation approach with efficient pruning rules.

9.6 Trajectory Mining on Diverse Applications

9.6.1 Social Networks and Social Media

Social networks and social media are prevalent on the Internet and have become an active research topic attracting many professionals and researchers from a variety of fields. By adding trajectory information, we can bring online social networks and media back to the physical world and share our real-life experiences in the virtual world conveniently. By mining trajectory patterns or predicting locations from social networks, people can not only track and share location-related information with each other via mobile devices or desktop computers, but can also leverage collaborative social knowledge learned from user-generated and location-related content. As trajectory is one of the most important properties in people's everyday lives, the research on trajectory mining in social networks bridges the gap between online societies and the physical world, and enable novel applications that have the potential to change the way we live, e.g., path planning/prediction, friend suggestion, location/friend recommendations, community discovery, human mobility modeling, and user activity analysis. In the chapter 8.3.2, we investigated that trajectories can be applied on tweets to better evaluate the behavior of communities. Conducted experiments were mainly defined to highlight that trajectories can be useful. It is clear that, according to the obtained results, trajectories are a new approach for better evaluating how behaviors from social networks can be analyzed. This point could be much more investigated.

9.6.2 Remote Sensing, Spatial Information on Satellite Image Processing

The GEOSUD (*GEOInformation for SUstainable Development*) inter-institutional pole of competence in remote sensing and spatial information has been created and the GEOSUD project has been launched in 2011, as the result of a close collaboration between fourteen institutional partners. Coordinated by Irstea, the project offers online free access to homogeneous and up-to-date spatial information on French ecosystems and territories. It aims at promoting networking between the scientific community and both public and private actors in land management, addressing to researchers and decision makers [1]

1. http://www.peer.eu/news-events/detail/?tx_list_pi1%5Buid%5D=153

Figure 9.4: Left: Subset of a Quickbird satellite image of Rostock (© DigitalGlobe, Inc., 2011), right: derived land cover classes.

Indeed, satellite images provide useful data to understand and manage territories by developing geo-information methods to be applied to the management of forests, water, cultivated lands, biodiversity, and natural hazards. Today with the rapid development of new satellite sensors we are provided with satellites providing high-resolution images. For instance, Quickbird, SPOT-5, OrbView, GeoEye, Pléiades, etc. These sensors typically have a panchromatic band of finer spatial resolution ($\sim$ 0.5 to 2 m) and 3 or 4 less accurate multi-spectral bands ($\sim$ 1.6 to 5 m). Using these provided images over time can be very useful to detect the evolution of objects. For instance in the timelapse project, Google and the Nasa have recently reported evolution of some places in the world by three decades of satellite images [2].

Let's consider the Figure 9.4, we have the satellite image on the left hand side and the right hand side image presents the land cover objects which are a result of an object-based image segmentation and classification [3]. In order to learn the evolution of objects by the time, we slit the image into pixel grid such that each pixel will be labeled as a part of a

2. http://world.time.com/timelapse/

3. Image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an

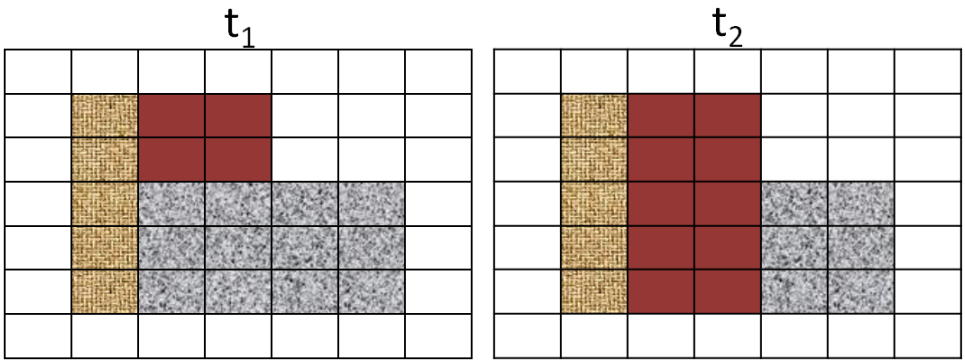


Figure 9.5: An example of trajectory pattern mining on satellite images.

specific object. Let us consider the Figure 9.5, we have 49 pixels and there are three meaningful objects presented in different colors varying in two timestamps t_1 and t_2. Then, if each pixel plays role as a transaction and each object could be considered as a cluster, our proposed approaches, e.g. Incremental GeT_Move, ClusterGrowth, CoClusterGrowth, etc., can be directly applied to extract trajectory patterns such as convoys, closed swarms, moving clusters, gradual trajectories which are very useful for Geography scientist to understand and manage territories.

image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics. The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image.

CHAPTER

10 Publications

10.1 International Conferences and Journals

1. **Nhat Hai Phan**, Dino Ienco, Pascal Poncelet, Maguelonne Teisseire. *"Mining Representative Movement Patterns through Compression"*. The 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (**PAKDD 2013**), Goal Coast, Australia, April 2013.
2. **Nhat Hai Phan**, Dino Ienco, Pascal Poncelet, Maguelonne Teisseire. *"Mining Fuzzy Moving Object Clusters"*. In Proceedings of the 8th International Conference on Advanced Data Mining and Applications (**ADMA**), Nanjing China, December 2012.
3. **Nhat Hai Phan**, Dino Ienco, Pascal Poncelet, Maguelonne Teisseire. *"Mining Time Relaxed Gradual Moving Object Clusters"*. In Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (**ACM GIS 2012**), Redondo Beach, California, November 2012.
4. F. Bouillot, **Nhat Hai Phan**, N. Béchet, S. Bringay, D. Ienco, S. Matwin, P. Poncelet, M. Roche, and M. Teisseire. *"How to Extract Relevant Knowledge from Tweets?"*. The 7th International Workshop on Information Search, Integration and Personalization (**ISIP'2012**), Sapporo, Japan, October 2012.
5. **Nhat Hai Phan**, Pascal Poncelet, Maguelonne Teisseire. *"*GET_MOVE*: An Efficient and Unifying Spatio-Temporal Pattern Mining Algorithm for Moving Objects"*. In Proceedings of the 11th International Symposium on Intelligent Data Analysis (**IDA 2012**), Helsinki, Finland, October 2012.
6. **Nhat Hai Phan**, Pascal Poncelet, Maguelonne Teisseire. *"An Efficient Spatio-Temporal Mining Approach to Really Know Who Travels with Whom!"*. In 28th Advance in Data Mining (**BDA 2012**), Clermont Ferrand, France, October 2012. **(selected as Best papers)**

7. **Nhat Hai Phan**, Dino Ienco, Pascal Poncelet, Maguelonne Teisseire. *"Extracting Trajectories through an Efficient and Unifying Spatio-Temporal Patten Mining System"*. In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (**ECML-PKDD 2012**), Demo Paper, Bristol, UK, September 2012.

8. **Nhat Hai Phan**, Pascal Poncelet, Maguelonne Teisseire. *"Moving Objects: Combining Gradual Rules and Spatio-Temporal Patterns"*. 2011 International Conference on Spatial Data Mining and Geographical Knowledge Services (**ICSDM 2011**), Fuzhou, China, June 2011.

9. **Nhat Hai Phan**, Pascal Poncelet, Maguelonne Teisseire. *"An Efficient Spatio-Temporal Mining Approach to Really Know Who Travels with Whom!"*. Ingénierie des Systèmes d'Information (**ISI special issue, selected papers from BDA'12**), 2013, to appear.

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94*, pages 487–499, 1994. Cited pages 11, 20, and 71.

[2] H. H. Aung and K. L. Tan. Discovery of evolving convoys. In *SSDBM*, pages 196–213, 2010. Cited pages 7, 17, 19, and 21.

[3] R. Cilibrasi and P. M. B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005. Cited page 86.

[4] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231, 1996. Cited pages 16, 17, 19, 34, 63, 80, and 111.

[5] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. *TKDD*, 1(3), 2007. Cited page 86.

[6] P. Grunwald. The minimum description length principle. *The MIT Press*, 2007. Cited pages 72 and 100.

[7] J. Gudmundsson and M. V. Kreveld. Computing longest duration flocks in trajectory data. In *ACM GIS*, pages 35–42, 2006. Cited pages 7, 11, 19, and 62.

[8] P. N. Hai, D. Ienco, P. Poncelet, and M. Teisseire. Extracting trajectories through an efficient and unifying spatio-temporal pattern mining system. In *ECML/PKDD (2)*, pages 820–823, 2012. Cited pages 34, 63, 80, 81, and 96.

[9] P. N. Hai, D. Ienco, P. Poncelet, and M. Teisseire. Mining fuzzy moving object clusters. In *ADMA*, pages 100–114, 2012. Cited pages 7 and 21.

[10] P. N. Hai, D. Ienco, P. Poncelet, and M. Teisseire. Mining time relaxed gradual moving object clusters. In *ACM SIGSPATIAL GIS*, pages 478–481, 2012. Cited pages 21, 67, 96, 98, 110, and 111.

[11] P. N. Hai, D. Ienco, P. Poncelet, and M. Teisseire. Mining representative movement patterns through compression. In *PAKDD*, 2013, to appear. Cited pages 21, 86, and 87.

[12] P. N. Hai, P. Poncelet, and M. Teisseire. Get_move: An efficient and unifying spatio-temporal pattern mining algorithm for moving objects. In *IDA*, pages 276–288, 2012. Cited pages 10, 21, 56, 60, 62, 80, and 96.

[13] P. N. Hai, P. Poncelet, and M. Teisseire. An efficient spatio-temporal mining approach to really know who travels with whom! In *BDA 2012*, Clermont-Ferrand, France, 2012. Cited pages 56 and 60.

[14] J. Han, Z. Li, and L. A. Tang. Mining moving object, trajectory and traffic data. In *Database Systems for Advanced Applications*, volume 5982 of *Lecture Notes in Computer Science*, pages 485–486. 2010. Cited pages 8, 17, and 19.

[15] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000. Cited pages 11, 20, and 71.

[16] Y. Huang, S. Shekhar, H. Xiong. Discovering collocation patterns from spatial data sets: a general approach. *IEEE Transactions on Knowledge and Data Engineering* 16(12), pages 1472–1485, 2004. Cited page 57.

[17] C.S. Jensen, L. Dan, and B. C. Ooi. Continuous clustering of moving objects. *TKDE*, 19(9):1161 –1174, sept. 2007. Cited pages 7, 11, and 71.

[18] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *Proc. VLDB Endow.*, 1(1):1068–1080, August 2008. Cited pages 7, 11, 17, 19, 21, 34, 63, 71, 80, 96, and 111.

[19] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD '05*, pages 364–381, 2005. Cited pages 7, 17, 19, 21, 63, and 71.

[20] E. J. Keogh, S. Lonardi, C. A. Ratanamahatana, L. Wei, S. H. Lee, and J. Handley. Compression-based data mining of sequential data. *Data Min. Knowl. Discov.*, 14(1):99–129, 2007. Cited page 86.

[21] H. T. Lam, F. Moerchen, D. Fradkin, and T. Calders. Mining compressing sequential patterns. In *SDM*, pages 319–330, 2012. Cited pages 86, 87, 88, and 103.

[22] J. G. Lee, J. Han, and K. Y. Whang. Trajectory clustering: a partition-and-group framework. In *ACM SIGMOD '07*, pages 593–604, 2007. Cited page 21.

[23] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: mining relaxed temporal moving object clusters. *Proc. VLDB Endow.*, 3(1-2):723–734, September 2010. Cited pages 7, 11, 16, 17, 20, 21, 34, 37, 55, 59, 62, 63, 64, 71, 80, 96, 110, and 111.

[24] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and Da. W. Cheung. Mining, indexing, and querying historical spatio-temporal data. In *KDD '04*, pages 236–245, 2004. Cited pages 7 and 18.

[25] R. Milo, S. Shen-Orr, S. Itzkovits, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594), 2002. Cited page 86.

[26] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995. Cited page 90.

[27] K. Smets and J. Vreeken. Slim: Directly mining descriptive patterns. In *SDM*, 2012. Cited pages 87 and 88.

[28] L. A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C.-C. Hung, and W.-C. Peng. On discovery of traveling companions from streaming trajectories. In *ICDE*, pages 186–197, 2012. Cited pages 7, 20, and 21.

[29] F. Verhein. Mining complex spatio-temporal sequence patterns. In *SDM '09*, pages 605–616, 2009. Cited page 7.

[30] M. R. Vieira, P. Bakalov, and V. J. Tsotras. On-line discovery of flock patterns in spatio-temporal data. In *ACM SIGSPATIAL GIS*, pages 286–295, 2009. Cited pages 7, 19, and 21.

[31] J. Vreeken, M. Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Min. Knowl. Discov.*, 23(1):169–214, 2011. Cited pages 72, 86, 88, 100, and 104.

[32] Y. Wang, E. P. Lim, and S. Y. Hwang. Efficient mining of group patterns from user movement data. *Data Knowl. Eng.*, 57(3):240–282, June 2006. Cited pages 7, 18, 20, 37, 63, 71, and 80.

[33] Z. Li, M. Ji, J.-G. Lee, L.-A. Tang, Y. Yu, J. Han, and R. Kays. MoveMine: mining moving object databases. In *ACM SIGMOD '10*, pages 1203–1206, 2010. Cited pages 12, 34, and 144.

[34] A. Knobbe, and E. Ho. Pattern teams. In *PKDD '06*, pages 577–584, 2006. Cited page 87.

[35] B. Bringmann, and A. Zimmermann. The chosen few: On identifying valuable patterns. In *ICDM '07*, pages 63–72, 2007. Cited page 87.

[36] A. Siebes and R. Kersten. A structure function for transaction data. In *SDM '11*, pages 558–569, 2011. Cited page 87.

[37] H. Mannila, E. Terzi. Nestedness and Segmented Nestedness. In *KDD'07*, San Jose, California, USA, pages 480–489, 2007. Cited pages 2, 42, and 45.

[38] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 2004), volume 126 of CEUR Workshop Proceedings, Brighton, UK, 1 November 2004. Cited page 29.

[39] A. O. C. Romero. Mining moving flock patterns in large spatio-temporal datasets using a frequent pattern mining approach. *Master Thesis*, University of Twente, faculty ITC, March 2011. Cited page 7.

[40] A. Appice, M. Ceci, and D. Malerba. Mining model trees: A multi-relational approach. In *Proc. of the 13th International Conference on Inductive Logic Programming, ILP 2003*, volume 2835 of *LNAI*, pages 4–21. Springer-Verlag, 2003. Cited page 116.

[41] H. Mannila B. Goethals, and W. L. Page. Mining association rules of simple conjunctive queries. In *SDM 2008*, pages 96–107, 2008. Cited page 138.

[42] T. Calders, B. Goethals, and S. Jaroszewicz. Mining rank-correlated sets of numerical attributes. In *KDD 2006*, pages 96–105. Cited pages 116, 117, 118, 121, and 123.

[43] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Min. Knowl. Discov.*, 3(1):7–36, March 1999. Cited pages 116 and 138.

[44] L. Di-Jorio, A. Laurent, and M. Teisseire. Mining frequent gradual itemsets from large databases. In *IDA 2009*, pages 297–308. Cited pages 116, 117, 118, 121, 123, 127, 128, 130, and 131.

[45] L. Džeroski and N. Lavrač. *Relational data mining*. Springer-Verlag, Berlin, 2001. Cited page 116.

[46] P.A. Flach and N. Lachiche. Naive bayesian classification of structured data. *Machine Learning*, 57(3):233–269, 2004. Cited page 116.

[47] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In Thomas Dean, editor, *IJCAI*, pages 1300–1309. Morgan Kaufmann, 1999. Cited page 116.

[48] B. Goethals, W. L. Page, and M. Mampaey. Mining interesting sets and rules in relational databases. In *SAC 2010*, pages 997–1001, 2010. Cited page 138.

[49] E. Hullermeier. Association rules for expressing gradual dependencies. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, volume 2431 of *Lecture Notes in Computer Science*, pages 200–211. Springer Berlin Heidelberg, 2002. Cited page 116.

[50] A. Koopman and A. Siebes. Characteristic relational patterns. In *KDD 2009*, pages 437–446. Cited pages 119, 120, and 138.

[51] A. Koopman and A. Siebes. Discovering relational item sets efficiently. In *SDM 2008*, pages 108–119, 2008. Cited page 138.

[52] E. K. K. Ng, A. W.-C. Fu, and K. Wang. Mining association rules from stars. In *ICDM 2002*, pages 322 – 329, 2002. Cited page 138.

[53] S. Nijssen and J. Kok. Efficient frequent query discovery in farmer. In *PKDD 2003*, pages 350–362, 2003. Cited page 138.

[54] W.L. Page. Mining patterns in relational databases. In *PhD Thesis, Universiteit Antwerpen, 2009*. Cited pages 119, 120, and 138.

[55] E. Spyropoulou and T. D. Bie. Interesting multi-relational patterns. In *ICDM 2011*, pages 675–684, 2011. Cited pages 119 and 138.

[56] F. Zelezný and N. Lavrac. Propositionalization-based relational subgroup discovery with rsd. *Machine Learning*, 62(1-2):33–63, 2006. Cited page 116.

[57] S. Ayouni, A. Laurent, S. B. Yahia, and P. Poncelet. Mining Closed Gradual Patterns. In *ICAISC (1)*, pages 267-274, 2010. Cited pages 117 and 118.

[58] M. J. Zaki. Mining Non-Redundant Association Rules. *Data Min. Knowl. Discov.*, 9(3):223–248, 2004. Cited page 29.

[59] S. Rinzivillo, D. Pedreschi, M. Nanni, F. Giannotti, N. Andrienko, and G. Andrienko. Visually driven analysis of movement data by progressive clustering. *Information Visualization*, 7(3):225–239, 2008. Cited page 21.

[60] G. L. Andrienko and N. V. Andrienko. Spatio-temporal aggregation for visual analysis of movements. In *IEEE VAST*, pages 51–58, 2008. Cited page 21.

[61] G. L. Andrienko and N. V. Andrienko. Interactive cluster analysis of diverse types of spatiotemporal data. *SIGKDD Explorations*, 11(2):19–28, 2009. Cited page 21.

[62] G.L. Andrienko, N.V. Andrienko, S. Rinzivillo, M. Nanni, D. Pedreschi, and F. Giannotti. Interactive visual clustering of large collections of trajectories. In *IEEE VAST*, pages 3–10, 2009. Cited page 21.

[63] A.T. Palma, V. Bogorny, B. Kuijpers and L.O. Alvares. A clustering-based approach for discovering interesting places in trajectories. In *ACM SAC '08*, pages 863–868, 2008. Cited page 21.

[64] J. H. Kang, W. Welbourne, B. Stewart and G. Borriello. Extracting places from traces of locations. In *Proceedings of the 2nd ACM international workshop on Wireless mobile applications and services on WLAN hotspots (WMASH '04)*, pages 110–118, 2004. Cited page 21.

[65] F. Giannotti, M. Nanni, F. Pinelli and D. Pedreschi. Trajectory pattern mining. In *ACM SIGKDD*, pages 330–339, 2007. Cited pages 20 and 21.

[66] F. Bouillot, P. Poncelet, M. Roche, D. Ienco, E. Bigdeli and S. Matwin. French Presidential Elections: What are the Most Efficient Measures for Tweets?. In Proceedings of the Workshop Politics, Elections and Data (PLEAD 2012) in conjunction with the 21st ACM International Conference on Information and Knowledge Management (CIKM 2012), Maui, USA, November 2012. Cited page 149.

Abstract

Mining object movement patterns to understand the behavior of moving objects has many high impact applications. However even if existing pattern models are meaningful, there are still challenging issues such as: 1) there are many kinds of patterns and algorithms without an efficient management scheme, 2) lacking of relevant approaches in dealing with time gaps, 3) they usually focus on an unchanged group of objects and thus do not capture the behavior of the whole object class, 4) Naturally, there are huge amount of very redundant movement patterns extracted while only a few of them are meaningful. However, few researchers address this issue. 5) Despite the growing demands for diverse applications, there have been few scalable tools for mining massive and sophisticated moving object data.

In my thesis, I will mainly focus on addressing these issues. We propose the three step framework: 1) the first step aims to present and framework to mine and manage different existing movement patterns in an efficient way, 2) in the second step, we propose novel movement pattern concepts to access the relevance of movement patterns by dealing with time gaps. We also further present the *gradual trajectory pattern* notion to analysis the behavior of moving objects in a graduality point of view. 3) In the last step, we propose an novel MDL principal-based approach, named SmartCompo to extract representative movement patterns from moving object data. 4) Our three step framework is illustrated in a demonstration system, named *Multi_Move*, which is designed to extract and manage different kinds of spatio-temporal patterns concurrently. A user-friendly interface is provided to facilitate interactive exploration of mining results.

As an extra work, I further present the concept of *multi-relational gradual pattern* which generalizes the gradual pattern notion in single relation data to multi-relation database.

Keywords: *Moving Object, Object Movement Pattern, Compressing Movement Patterns, Multi-Relational Gradual Patterns, Gradual Trajectory*

LIRMM — 161, rue Ada — 34095 Montpellier cedex 5 — France