Efficient mining of group patterns for mobile users

Wang, Yida

2005

Wang, Y. (2005). Efficient mining of group patterns for mobile users. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2526>

<https://doi.org/10.32657/10356/2526>

Nanyang Technological University

Downloaded on 26 Apr 2023 19:58:13 SGT

NANYANG TECHNOLOGICAL UNIVERSITY

Efficient Mining of Group Patterns for Mobile Users

Wang Yida

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of

Doctor of Philosophy

2005

Abstract

Mobile phones and other mobile devices are fast becoming indispensable in our modern society, inspiring many emerging applications. In this thesis, we propose novel approaches to derive useful *grouping* information from a movement database of users' locations gathered from mobile devices over a time period. We are interested in discovering groups of users such that members in the same group are spatially close to one another for significant amount of time. Such a user group is also known as *group pattern*.

In order to discover the complete set of valid group patterns, we develop two algorithms, namely AGP and VG-growth. AGP is derived from the well known Apriori algorithm [AS94], while VG-growth adopts a mining strategy similar to FP-growth algorithm [HPY00] but is based on a novel data structure known as *VG-graph*. We have conducted extensive experiments to evaluate the performance of AGP and VG-growth using datasets synthetically generated by IBM City Simulator [KMJ01]. The results have shown that VG-growth is much faster than AGP for mining valid k -groups ($k > 2$), especially for small *min_weight* threshold.

As the mining of valid groups of size two is a bottleneck for both AGP and VG-growth, especially for large number of users and long logging duration, we further propose a group pattern mining framework that accommodates different location summarization methods. Each method reduces the number of time points to be scanned and the number of candidate 2-groups to be examined. Under this framework, several location summarization methods have been proposed. Experiments have shown that the location summarization based algorithms reduce the mining overhead for valid group patterns of size two significantly.

We finally identify the redundancy problem of mining the complete set of valid group patterns due to the exponential enumeration of all possible valid groups. We therefore address this problem by mining only the set of *maximal valid groups*, which provides the

complete and non-redundant information regarding all valid groups. We first develop algorithms AMG and VGMax, which are extensions of AGP and VG-growth respectively, to mine maximal valid groups. Next, we present algorithm VGBK derived from the Bron-Kerbosch algorithm [BK73] for enumerating all maximal cliques in an undirected graph. VGBK uses simple criteria to identify maximal valid group without maximality checking. The experimental results have shown that the proposed maximal valid group mining algorithms are effective and efficient.

Acknowledgments

I would like to express my gratitude to the following organizations and people for their support and contribution:

Nanyang Technological University, Singapore, for granting me the opportunity and scholarship to undertake the Ph.D. (Computer Engineering) programme and for providing a most conducive research environment in the Centre for Advanced Information Systems (CAIS) for my research activities.

Associate Professor Lim Ee-Peng, my supervisor, for his invaluable guidance, encouragement, understanding and sharing of his knowledge. As an advisor, he demonstrates knowledge and patience. He spends a lot of time to discuss issues, and leaves enough room for me to venture on my own. From him, I learned how to think systematically, deeply, and carefully; how to write and present my work clearly.

Professors, researchers and students at CAIS, for their generous sharing of ideas, suggestions, and discussions, as well as the help in reviewing my writings.

The technical staff at CAIS: Mr. Lai Chee Keong and Ms. Tan Lay Choo, who worked hard to ensure the smooth operations of our computers, printers and network.

Last but not least, my parents, my wife, and my sister, for their love, care, support and understanding that they have given me to pursue my interest in research.

Thank you all !

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vii
List of Tables	ix
List of Algorithms	x
1 Introduction	1
1.1 Motivation	1
1.2 Mining Group Patterns from Movement Data	2
1.3 Contribution	5
1.4 Thesis Outline	8
2 Related Work	10
2.1 Social Network Analysis	10
2.2 Positioning Technologies	13
2.2.1 Outdoor Environment	13
2.2.2 Indoor Environment	16
2.3 Frequent Itemset Mining	16
2.3.1 Preliminaries	16
2.3.2 Apriori Algorithm	18
2.3.3 Breaking the Apriori Bottleneck	18
2.4 Time Series Analysis	22
2.5 Sequential Pattern Mining	25
2.6 Mining Spatial or Spatio-Temporal Data	26
2.7 Groups in Mobile Computing	27

3	Group Pattern Mining	28
3.1	Problem Definition	28
3.2	Mining Valid Groups	32
3.2.1	AGP Algorithm	32
3.2.2	VG-growth Algorithm	37
3.3	Performance Evaluation	44
3.3.1	Experimental Design and Data Generation	44
3.3.2	Performance Evaluation	45
3.4	Summary	48
4	Efficiently Mining Valid 2-Groups by Location Summarization	50
4.1	Framework for Mining Valid 2-Groups Using Location Summarization . . .	50
4.1.1	Preprocessing of User Movement Data	51
4.1.2	Mining Valid 2-Groups	55
4.2	Location Summarization Methods	58
4.2.1	Sphere Location Summarization (SLS) Method	58
4.2.2	Cuboid Location Summarization (CLS) Method	59
4.2.3	Grid-Sphere Location Summarization (GSLS) Method	60
4.2.4	Grid-Cuboid Location Summarization (GCLS) Method	62
4.2.5	Summary of Location Summarization Methods	63
4.3	Performance Evaluation of Location Summarization Based Algorithms . . .	63
4.3.1	Experimental Design and Data Generation	63
4.3.2	Performance Evaluation	64
4.4	Summary	70
5	Maximal Valid Group Mining	72
5.1	Problem Definition	72
5.2	Related Work	73
5.2.1	Mining Maximal Frequent Itemsets	73
5.2.2	Maximal Clique Enumeration	76
5.3	Extensions of AGP and VG-growth for Mining Maximal Valid Groups . . .	76
5.3.1	AMG Algorithm	76
5.3.2	VGMax Algorithm	77

5.4	Mining Maximal Valid Groups by Finding Maximal Cliques	83
5.4.1	Bron-Kerbosch Algorithm	83
5.4.2	VGBK Algorithm	85
5.5	Performance Evaluation	90
5.5.1	Experimental Design and Data Generation	90
5.5.2	Performance Comparison	91
5.6	Summary	96
6	Conclusion and Future Work	98
6.1	Summary of the Thesis	98
6.2	Future Work	100
A	GSLs and GCLS Methods with Maximum Speed Constraint	104
B	Parameter Settings for Grid Based Location Summarization	107
C	List of Publications	110
	References	111

List of Figures

2.1	An Example of Social Network	11
3.1	Search Tree for AGP	36
3.2	The VG-graph for Table 3.1	38
3.3	Search Tree for VG-growth	42
3.4	An Illustrative Example of VG-growth	43
3.5	Experiment Results: AGP vs. VG-growth	46
3.6	Experiment Results: Scalability of VG-growth	48
4.1	Example of Instances of Summarization Model	51
4.2	SLS vs. CLS	59
4.3	Partition 3D Space into Cells	61
4.4	Impact of time window size	66
4.5	Impact of $\overline{max_dis}$	67
4.6	Performance Comparison with Fixed $ D' $	69
4.7	Performance Comparison with AGP	70
5.1	An Set Enumeration Tree over Four Items	74
5.2	Search Tree for AMG	78
5.3	An Illustrative Example for VGMax	79
5.4	Search Tree for VGMax	81
5.5	Search Tree for VGBK	87
5.6	Different Conditional VG-graphs used by VGBK and VG-growth	89
5.7	Execution Time Components	92
5.8	Results: Dataset M1kN1k (min_wei changes)	93
5.9	Results: Dataset M1kN1k (max_dis or min_dur changes)	94

5.10 Results: Dataset M6kN1k	95
A.1 Illustration of GCLS2 Methods	105

List of Tables

3.1	User Movement Database D	29
3.2	Closeness between Users in Table 3.1 ($max_dis = 10$)	30
3.3	Parameter Setting: AGP vs. VG-growth	45
3.4	Parameter Setting: Scalability of VG-growth	47
4.1	Summary of Location Summarization Methods	63
5.1	(Maximal) Valid Groups Involved in Table 3.1	73
5.2	Complete Mining Process of VGBK on Figure 3.2	88
5.3	Experimental Settings	91
B.1	Parameter Settings for $\frac{ D' }{ D } = \frac{1}{120}$ (0.83%)	108
B.2	Parameter Settings for $\frac{ D' }{ D } = \frac{1}{12}$ (8.33%)	109

List of Algorithms

2.1	Apriori Algorithm for Finding Frequent Itemsets	19
3.1	AGP Algorithm.	33
3.2	Functions used by AGP Algorithm.	34
3.3	VG-growth Algorithm.	41
4.1	Preprocessing Step of Framework	55
4.2	Mining Step of Framework	56
5.1	AMG Algorithm.	77
5.2	VGMax Algorithm.	82
5.3	Bron-Kerbosch Algorithm (Version 1).	84
5.4	VGBK Algorithm.	86

Chapter 1

Introduction

1.1 Motivation

Mobile phones and other similar devices are fast becoming indispensable in our modern society. According to a recent survey by Frank N. Magid Associates and Upoc.com, 59 percent of Americans age 12 and over (about 140 millions of them) own mobile phones, and that almost a quarter of non-owners plan to buy a mobile phone in the near future [Up03]. The sales of mobile phones worldwide was 385 millions in 2001 and it has been predicted to reach 675 million in 2006 [Res02].

In tandem with this growth trend, we also witness the emergence of many new applications and businesses that exploit mobile phone technologies in different ways [VVK00]. Mobile phones, unlike computers connected to wired networks, are highly *personalized*. While it is common for a user to own a few mobile phones, it is very unlikely for different users to share a mobile phone. That is, the physical location of a mobile phone probably represents the location of its owner. Also unlike other personalized accessories such as watches, walkmans, etc., mobile phones are *trackable*. They are trackable because they have to maintain regular contacts with the mobile telecommunication networks in order to receive and make calls. Modern positioning technologies, such as GPS [Zar96], with a wide range of accuracies have been popularly used in many applications. With these trackability and personalized features, it is possible to collect the location data, or movement data, for mobile phone users.

In this thesis, we propose several novel approaches to derive useful *grouping* information from movement data of mobile users. What is a group? Although definitions vary among theorists, most definitions are consistent with that of Forsyth [For99], who defines a group

as “two or more interdependent individuals who influence one another through social interaction”. People form groups due to many different reasons. Within an organization, people are assigned into different groups to carry out some designated tasks. On the other hand, more social groups are formed due to kindred, emotional, or psychological needs. Group dynamics and its influence on individual decision making have been well studied by sociologists [Fis93, Lev94, HCH94, Cas95, For96, For99], and it has been shown that peer pressure and group conformity can affect the behaviors of individuals. Such group behaviors can be very useful in many applications as we will see later.

In the past, different ways to discover groups using clustering techniques have been proposed [SKR01, YP03, She04]. Very often, they are based on different definitions of similarity measure to represent the closeness between users. For example, users may be grouped by their common interests, job features, education level, and other attributes. Users can also be grouped based on the transactions they perform. For example, Amazon.com groups users together by the common books they purchase. However, in many cases, these methods suffer from a common pitfall, i.e., members in a derived group may not even know one another. Such kind of group derivation approaches are therefore not suitable to many applications that require group members to be acquaintances.

1.2 Mining Group Patterns from Movement Data

In this research, we propose a new way to derive groups by mining the user movement log data. Several questions can be posed for this research.

What kinds of groups can be found from movement data?

We are interested in discovering *groups of users such that members in the same group are spatially close to one another for significant amount of time*. Intuitively, people tend to maintain close physical distance with other persons with whom they have certain relationships. For example, classmates take courses in the same lecture room in weekdays, team members often play basketball in the court, shopping pals shop around together in weekends, etc.. The grouping knowledge derived in this way is unique compared to the other approaches due to the following:

- *Physical proximity between group members*

The group members are expected to be physically close to one another when they act as a group. Such characteristics are common among many types of groups, e.g., shopping pals, game partners, etc..

- *Temporal proximity between group members*

The group members are expected to stay together for some meaningful duration when they acts as a group. Such characteristics distinguishes an ad hoc cluster of people who are physically close but unaware of one another, e.g., the passengers staying on the same bus during a short time, from a group of people who come together for some planned activity(ies).

Intuitively, people who spent significant time together are expected to be *aware* of one another and they should maintain regular *contact*. Hence, the group members are expected to exert much stronger influence on one another.

What are the necessary assumptions?

The movement data are assumed to be generated by mobile devices that track the locations of their owners as they move from one place to another by using positioning technologies such as GPS [Zar96]. We also assume that each user location, in the form of x-/y-/z- coordinates, can be logged at regular intervals over a period of time. In practice, the assumption may not hold as mobile devices may experience failures. They may be switched off by their owners from time to time, and the data collection time may not be synchronized across users. These assumptions nevertheless are reasonable if considering data cleaning or data transformation to be performed before applying our proposed mining algorithms. Although logging user locations over time can cause privacy-invasion problems, to keep a focused discussion, we shall keep the privacy and legal issues out the scope of this thesis. We believe that these issues should be addressed within a legal framework which is beyond the scope of this thesis. Furthermore, for several practical situations related to safety and security, tracking user movement is considered necessary and has already been done [Hoo04]. For example, people may be willing to know the locations where their young children or grandparents are so as to act immediately when they are in trouble [Feh04].

What are the algorithms to find these groups?

This challenge refers to the algorithms for mining group patterns from a large movement database. There are several concerns here. Firstly, the algorithms should generate all the expected group patterns, i.e., meeting the *completeness* requirement. Secondly, the algorithms should only generate the expected group patterns, i.e., meeting the *correctness* requirement. Thirdly, as movement databases normally contain large population and long sampling duration, the *efficiency* of algorithm is very important and this involves two aspects: space efficiency and time efficiency.

How can the discovered group patterns be verified?

It is important to distinguish the correctness of the mining results from the correctness of the mining algorithms. The key difference between them is the different evaluation baseline. The former is verified against *the set of social groups formed in real life*, while the latter is evaluated based on *the definition of group pattern*.

After discovering the group patterns, we need to verify whether the discovered group patterns are truly correct with respect to the social groups found in real life. To achieve this, one possible way is to adopt the standard *recall* and *precision* measures. Recall measures how many groups that are supposed to be discovered have been correctly discovered. Precision, on the other hand, measures how accurate the discovered results are, i.e., is there any discovered group that is in fact not a real social group. In order to use recall and precision measures, we have to identify the set of real social groups, which act as a baseline to verify the groups derived based on the spatio-temporal relationship. To obtain the set of real social groups, questionnaires or interviews may have to be conducted on the users. This exercise may be costly if the number of users involved are large.

How can the discovered groups be used in real life applications?

The discovered group patterns can be very useful in many applications. For example, by knowing the groups a customer belongs to, retailers can derive common buying interests among customers and develop group-specific pricing models and marketing strategies. Group discounts and product recommendation can be introduced to encourage more purchases that lead to higher sales with better success rate. In fighting against terrorism, analyzing user group patterns is one of the important tasks that help to reveal the links

between terrorists and their roles in the group. In addition, group information could be very important to locate the sources of a highly infectious disease, such as SARS.

By combining with the location information, group pattern mining can find out not only who belongs to a group but also the places where the group members often visit. Coupled with the timing information, group patterns can convey very rich knowledge about the activities of group members.

The group pattern mining techniques proposed in this thesis is also not confined to mobile users. It can be used in many other fields. For example, it can be used by biologists to analyze the group behavior of endangered animals. We expect in the future this technique could be used in broad spectrum of activities, personal and public, commercial and educational, business and pleasure.

1.3 Contribution

Our research aims to efficiently derive complete grouping information from large movement database for mobile users. We summarize our main research contributions as follows.

Valid group pattern mining as a new data mining problem

In this thesis, we formally define *group pattern*. We introduce *max_dis* and *min_dur* as the maximum physical distance threshold among group members and the minimum duration threshold for group members to stay together respectively for deriving group patterns. Furthermore, we define the *weight* of a group pattern as a measure of its interestingness. With a *min_wei* threshold, we define the notion of *valid group pattern* and the valid group pattern mining problem.

Algorithms for mining the complete set of valid groups

We observe that group pattern mining has not been addressed by the existing data mining research. Although there are some similarities between group pattern mining and classical frequent itemset mining, a few crucial challenges exist:

- Unlike frequent itemset mining that establishes associations between items that are likely to be purchased in the same transaction, group pattern mining deals with time series of user location information where the transaction concept does not exist. Thus the traditional transaction-based algorithms can not be directly applied here.

- The additional spatial and temporal constraints, represented by *max_dis* and *min_dur* thresholds, make the group pattern mining more complex than frequent itemset mining, where the only concern is whether an item presents in a transaction.
- The *weight* measurement is more complicated than *support* used in frequent itemset mining. The former in fact involves a list of time intervals whose lengths are larger than *min_dur*. Each time we update the weight, we have to update every time interval and remove those whose lengths are less than *min_dur*, instead of simple arithmetic increment/decrement, which is the case of support.

We therefore develop two algorithms, namely AGP and VG-growth, to mine valid group patterns. AGP is derived from the well known Apriori association rule mining algorithm [AS94]. It employs a breadth-first strategy, i.e., generating candidate valid groups of size k from valid groups of size $k - 1$ and scanning the movement database at each level to check the validity for each candidate valid group. To reduce overhead, AGP can first load the user movement database into the main memory to avoid multiple database scans on the hard disk. This alternative however cannot be applied to very large user movement databases with a limited memory size. The VG-growth algorithm overcomes the above memory constraint by scanning the movement database only *once* to construct a novel data structure known as *VG-graph* for further mining. VG-growth adopts a mining strategy similar to FP-growth algorithm [HPY00] but is based on VG-graph, a new data structure developed in this research.

We have conducted comprehensive experiments to evaluate the performance of AGP and VG-growth using datasets synthetically generated by IBM City Simulator [KMJ01]. We observed that VG-growth is much faster than AGP for mining valid k -groups, where $k > 2$, especially for small *min_wei* thresholds.

Breaking the bottleneck of mining valid 2-groups

We observed in the experiments that the time taken by AGP and VG-growth to mine valid 2-groups dominates the total mining time, because both algorithms require large number of candidate valid 2-groups to be examined, especially when the number of users is large. For example, given M users, there will be $\binom{M}{2}$ candidate valid groups to be considered.

In order to break such a bottleneck, we therefore propose a group pattern mining framework that can accommodate different location summarization methods. The main idea

is, before mining valid groups using either AGP or VG-growth, to first partition the user movement database into multiple time windows of equal size, and summarize the location information of every user in each time window by a summarization model. The objective here is to reduce the number of time points to be examined during the mining process. To further reduce the mining overhead, we pre-compute the maximum possible weights and durations for each candidate valid 2-group based on the summarized movement data. Using both the maximum possible weights and durations, the number of candidate valid 2-groups can be significantly reduced. Under this framework, several location summarization methods have been proposed which adopt different summarization models considering the tradeoff between the summarized database size and the summarization accuracy.

We have conducted a series of experiments to evaluate the proposed location summarization based algorithms for mining valid 2-groups. The experiment results have shown that the location summarization based algorithms are much more efficient than AGP and VG-growth for mining valid 2-groups. We also make some observations about the existence of optimal time window size given a user movement database.

Reducing the redundancy of mining complete set of valid groups

Using VG-growth and the location summarization based algorithms, one can efficiently find the complete set of valid groups from a large movement database. However, the complete set of valid groups can be very large due to the exponential enumeration of all possible valid groups. For example, given a valid group of five users, say $\{u_1, u_2, u_3, u_4, u_5\}$, all of its proper subgroups (25 of them) will be identified as valid groups as well, i.e., $\{u_1, u_2\}$, $\{u_1, u_3\}$, ..., $\{u_4, u_5\}$, $\{u_1, u_2, u_3\}$, ..., $\{u_3, u_4, u_5\}$, $\{u_1, u_2, u_3, u_4\}$, ..., $\{u_2, u_3, u_4, u_5\}$. In general, a group G of size k has $\binom{k}{2}$ subgroups¹ of size 2, $\binom{k}{3}$ subgroups of size 3, ..., $\binom{k}{k-1}$ subgroups of size $k-1$. If G is a valid group, all of the $2^k - k - 2$ subgroups will also be valid. The mining of all valid groups not only causes significant valid group mining overhead, but also compromises the usefulness of mining results due to the large proportion of redundant valid groups. This problem is similar to mining *frequent itemsets* v.s. mining *maximal frequent itemsets* [Bay98].

We can address the above redundancy problem by mining only the set of maximal valid groups. Maximal valid group mining poses two challenges.

¹Please note that a single user is not considered to be a 1-group since a proper group must have at least two members.

- Firstly, the existing algorithms for mining maximal frequent itemsets are not applicable as the weight computation of valid groups is more complicated than support counting. This is due to a weight definition involving two thresholds *max_dis* and *min_dur* for determining valid segments.
- Secondly, most of the existing algorithms for mining maximal frequent itemsets require *maximality checking*, i.e., whenever a potential maximal frequent itemset is found, it is necessary to check whether it has a superset already found. This maximality checking can become costly in the later stage of the mining process because the set of maximal frequent itemsets found so far is ever growing during the mining process. The same problem will occur in mining the set of maximal valid groups if we directly apply these algorithms.

In this research, we propose several algorithms for mining maximal valid groups. We first develop two algorithms AMG and VGMax, which are extensions of AGP and VG-growth respectively. Next, we present algorithm VGBK that is derived from the Bron-Kerbosch algorithm [BK73] for enumerating all maximal cliques in an undirected graph. VGBK uses simple criteria to identify maximal valid group without maximality checking, which is normally a expensive operation. Our experimental results reveal that VGMax is more efficient for mining maximal valid groups when the total number of maximal valid groups is not too large while VGBK performs best when there are a large number of maximal valid groups.

1.4 Thesis Outline

The rest of the thesis is organized as follows.

- In Chapter 2, we conduct a brief literature survey on the topics related to this research.
- In Chapter 3, we formally define the valid group pattern mining problem and propose two algorithms for mining the complete set of valid groups.
- In Chapter 4, we introduce several location summarization based algorithms for efficiently mining valid groups of size two.

- In Chapter 5, we propose several algorithms for mining maximal valid groups to address the redundant group problem involved in valid group mining.
- In Chapter 6, we draw conclusions for this research.

Chapter 2

Related Work

We shall begin with an overview on some research topics related to our group pattern mining. The motivation for this is twofold. Firstly, the topics reviewed in this chapter will provide a better understanding of our discussion in the subsequent chapters. Secondly, the breath of the literature covered also indicates that the group pattern mining problem has not been addressed to the best of our knowledge.

2.1 Social Network Analysis

Group pattern mining attempts to derive grouping information of mobile users, which reveals social relationships among them. This is closely related to *social network* analysis, which focuses on patterns of relations among people [WF94].

In the past twenty years, social network analysis has been studied in sociology and communication science. In a social network, the people under consideration are called *actors* and the relations between them are called *ties*. A social network is normally modelled as a graph, where vertices represent actors and edges indicate ties. Edges can be associated with *weights* to indicate the strength. Normally, social network analysts are interested in describing networks of actors, finding important patterns from the networks, tracking the flow of information through actors, and discovering the effects these patterns and information flow have on the actors [GHW97].

Figure 2.1 gives an example of social network, where actors are names of famous movie actors, ties represent co-occurrence of pairs of actor names appearing in some webpages, and weights represent the frequencies of co-occurrences. For instance, the names “Al Pacino” and “Robert De Niro” appear in same webpages with a normalized frequency of 0.652, i.e.,

Figure 2.1: An Example of Social Network

in 65.2% of task relevant webpages. The percentage is very high as they are top actors who often worked together in movies including “The Godfather”.

A significant amount of social network research focuses on analyzing the influence of actors or the relationships among actors based on the constructed social networks [Fre92, CH00, KKT03, LC03, FMT04, LC04, MH04]. We briefly review some of them below.

- *Discovery of connection subgraphs.* In [FMT04], Faloutsos *et al.* addressed the problem of finding connection subgraphs from single relation social networks¹ which is too large to analyze. Given a pair of actors in the social network, the connection subgraph is defined as the subgraph of user specified size which conveys *most* information about the relationship between this pair of actors. The authors proposed a novel method to find the connection subgraphs by modeling the whole social network as an electrical circuit. That is, each edge represents a resistor whose conductance equal to the edge weight and thus the quantity of information can be indicated by the corresponding electrical current strength. Therefore, a connection subgraph should maximize the total electrical current flowing from the source actor to the destination actor.
- *Discovery of most influential nodes.* In [KKT03], Kempe *et al.* introduced the influence maximization problem, which aims to choose a set of most influential nodes from a given social network. That is, assume that actors have influence on one another. For example, actor *A* has influence on actor *B*, which has influence on another actor

¹Single relation social network is one that contains one type of relations.

C . Thus, C can be eventually influenced by A in an indirect way, i.e., through actor B . Given an initial set of actors, known as *target set*, the set of actors eventually influenced by the initial target set is known as *active set*. The task is to find a target set of given size which can result in the largest active set. A greedy algorithm has been proposed for this purpose.

Note that, the method reported in [KKT03] assumes that the extent to which the actors influence one another is known in advance, which may not be easy to achieve. On the other hand, results of group pattern mining also can reveal insight about the influence relationships among mobile users. Intuitively, users who participate in more valid groups of larger sizes should have stronger influence on others.

- Discovery of *novel links*. Lin *et al.* introduced the problem of finding novel links from social networks in [LC03]. Three types of novel link discovery problems are addressed: (1) *novel path discovery*, i.e., finding the most interesting path between two actors; (2) *novel loop discovery*, i.e., finding the most interesting loop starting and ending at a single actor; and (3) *significant nodes discovery*, i.e., finding the set of actors that are most significantly connected to a given actor. Note that, the interestingness of a path is measured by *rarity*. That is, the paths appear less frequently are considered more interesting and more significant, assuming that the whole social network describes the behavior of the general population. This work provides another point of view about interestingness of relations when analyzing social networks. In addition, the same authors also addressed some precursory issues of verifying the discovered results in an unsupervised mode [LC04].

It is important to note that social network analysis and group pattern mining are dealing with different problems. The former intends to find out patterns of relations among people, while the latter tries to find groups of people who maintain spatial closeness for significant amount of time. Furthermore, all of the social network analysis work mentioned above are carried out based on pre-defined social networks represented by graphs, while group pattern mining is conducted on large movement database involving spatio-temporal dimensions.

Generally, the social networks are assumed to be constructed in advance. The traditional way to construct social networks is through questionnaires, interviews, or observations [GHW97]. For example, a person may be asked to report about the people they go to for

advice on work matters and the people they go to for advice on personal matters. It is clear that such an effort would be prohibitively expensive and time consuming. For example, it may have taken a year for one to manually identify all the interactions in the networks for only two persons [Boi74]. Adamic *et al.* proposed a new way to extract social networks by analyzing the information on the Internet [AA03]. Based on an observation that personal homepages provide a glimpse into the social structure of university communities, they demonstrated a means of leveraging text, mailing list, and in and out-link information to construct social networks. Such web based method is more efficient than the traditional means, although the results of their method may be affected by some factors that are too general to be indicative of social connections, e.g., a common homepage structure consisting of the city name where the owner resides, or, a general institutional website such as <http://www.ntu.edu.sg>.

In fact, we can view group pattern mining as one possible method to construct a social network for a given population. This can be done by using the set of valid 2-groups found by group pattern mining algorithms. That is, for each valid 2-group we can construct a tie between the corresponding two members in a social network.

2.2 Positioning Technologies

In our research on group pattern mining, one basic assumption is that the movement database can be generated by collecting the location data of mobile devices over time using the positioning technologies, such as GPS. We therefore give a brief overview on the existing positioning technologies, which mainly fall into two categories: outdoor environment and indoor environment.

2.2.1 Outdoor Environment

The positioning technologies that are commonly used in outdoor environments include: Cell of Origin, Angle of Arrival, Time of Arrival, Time Difference of Arrival, and Global Positioning System.

Cell of Origin (COO)

Cell of Origin (COO) is also known as Cell of Identification (Cell-ID), which is the first widely used positioning technology in wireless network [SF99]. As indicated by the name,

it estimates the location of a mobile device using the cell area, or cell ID, since the network can determine which base station is serving at a given time. Therefore, the accuracy of COO generally depends on the size of the cell, which could range from 150 meters in radius for downtown areas to 30,000 meters in radius for rural areas [GKT02].

One advantage of COO is that no modifications to the existing networks and handsets are required. It can therefore be easily implemented for the existing mobile phone users. In [Zha02], the author suggested that COO should be implemented as a backup positioning method to provide approximate location information in case other more accurate methods fail to pinpoint the position.

Angle of Arrival (AOA)

Angle of arrival is also known as Direction of Arrival (DOA), which determines the location of mobile device based on *triangulation*. That is, one base station produces a directional straight line from it to the mobile device. A second directional straight line can be formed between another base station to the mobile device. Thus, the intersection of the two lines determines the location of the mobile device. The accuracy of AOA is generally in the range of 100 meters [HB01, ZGL03]. AOA requires more than one base stations and directional antennae or antenna arrays to measure the direction, making it difficult to embed AOA in mobile phones.

Time of Arrival (TOA) and Time Difference of Arrival (TDOA)

The TOA system determines the location of mobile device based on *trilateration*, i.e., plotting three circles and then finding the intersection point of them [Zha02]. Three base stations act as the centers of the three circles and the radii are determined by the distances between the base stations and the mobile device. The distances are computed by multiplying the speed of light to the propagation time of the radio wave from the base stations to the mobile device. TOA uses the absolute propagation time of a radio wave traveling between a base station and a mobile device, which requires the precisely synchronized clocks. Alternatively, we can measure the round-trip time of a signal transmitted between a source and a destination, leading to a result twice that of the one-way measurement. This does not require synchronization between the source and the receiver and is the most common means of measuring propagation time [ZGL03].

The TDOA system determines the mobile device's position by using time difference measurements instead of absolute propagation time as TOA does. That is, TDOA measures the arrival time difference of a signal emitted from two base stations. The set of points that has the same time difference, and thus the same distance difference, results in a hyperbola with the two base stations as foci. Another pair of base stations can determine a second hyperbola. The intersecting point of the two hyperbolic loci gives the location of the mobile device. Therefore, TDOA utilizes at least three base stations for positioning. The accuracy of TOA and TDOA is generally between 50 and 125 meters.

Global Positioning System

Global Positioning System, more commonly known as GPS, is a worldwide satellite-based radio navigation system. GPS system normally consists of: *space component*, *control component*, and *user component*. The space component is a constellation of 24 satellites, orbiting the earth at approximately 12,000 miles with period of 12 hours. The 24 satellites are arranged in six orbital planes, four satellites in each plane. At any time, there are at least four satellites that are visible from any place on the earth. The control component mainly monitors and adjusts the orbiting of satellites, and synchronizes their clocks. The user component refers to the GPS receiver, which can receive the signals transmitted from the satellites [Zar96, DR01, Zha02, MGJ02].

The GPS system pinpoints the location of the GPS receiver based on the same principle as TOA. That is, a GPS receiver receives radio signals transmitted from at least four satellites, the positions of which are known in advance. The distances between the receiver and the satellites can be determined by multiplying the propagation time by the speed of light. Therefore, at least four hemispheres can be formed with the satellites as the center and the distances as the radii. The intersection point, in the form of 3D — latitude, longitude, and altitude, is just where the receiver is.

Based on different GPS receivers, the accuracy of GPS system ranges from 100 meters to semi-meter or even less [CK00, GKT02]. As GPS receivers are becoming smaller and cheaper, GPS is increasingly used in applications such as traffic navigation, etc.. However, in order to operate properly, GPS receiver requires a clear view of the sky. Therefore, GPS system can not apply to indoor environment.

2.2.2 Indoor Environment

To the best of our knowledge, there is no widely accepted and used indoor positioning method so far. Most related research projects make their own location tracking system. However, the underlying principles are still similar to those in TOA/TDOA/GPS, e.g., obtaining the time (or distance) difference to some signal transmitters whose positions are known already. Typically, they use not only the radio signal, but also the ultrasonic signal [CK00, ZGL03]. Besides, in the past two years, increasing attention has been paid to radio frequency (RF) based techniques, which are particularly effective among the existing solutions because they provide ubiquitous coverage and use the inexpensive and popular IEEE 802.11 wireless LAN (WLAN) as the fundamental infrastructure [JYN05, CY05]. The WLAN and RF based methods can achieve accuracy around 1.5-3 meters.

2.3 Frequent Itemset Mining

In the subsequent sections, we will use the Apriori property for group pattern mining and extend the Apriori and FP-growth algorithms originally proposed for association rule mining. Hence, we present an overview of the existing research on association rule mining.

2.3.1 Preliminaries

Association rule mining was first introduced in 1993 by Agrawal, Imielinski and Swami [AIS93]. The formal statements of the association rule mining problem are bulleted below:

- Let the *universal itemset*, $\mathcal{I} = \{i_1, i_2, \dots, i_m\}$ be a set of literals called *items* and $|\mathcal{I}|$ be its size.
- Let $\mathcal{D}$ be a database of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$, and $|\mathcal{D}|$ be its size, i.e., the number of transactions.
- Each transaction T has an unique identifier, called *TID*.
- An *itemset* is an unordered set of unique items. An itemset containing k items is a *k-itemset*.
- For a given itemset $X \subseteq \mathcal{I}$ and a given transaction T , T *contains* X if and only if $X \subseteq T$.

- Let $s(X)$ be the *support* of an itemset X , which is the percentage of transactions in $\mathcal{D}$ that contains X .
- Let $|\mathcal{D}| \times s(X)$ be the *support count* of X , which is the number of transactions in $\mathcal{D}$ that contains X .
- Let min_sup be the *minimum support* threshold.
- An itemset X is *large* or *frequent* if $s(X) \geq \text{min_sup}$.
- An *association rule* is an implication of the form $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$.
- The association rule $X \Rightarrow Y$ holds in $\mathcal{D}$ with *support* $s(X \cup Y)$.
- The association rule $X \Rightarrow Y$ holds in $\mathcal{D}$ with *confidence* c if $c\%$ of the transactions in $\mathcal{D}$ that contains X also contains Y .
- Let min_conf be the *minimum confidence* threshold.
- Given $\mathcal{D}$, min_sup and min_conf , the problem of *mining association rules* is to discover all *strong* association rules that have support and confidence no less than min_sup and min_conf respectively.
- A *candidate itemset* is an itemset that is potentially a frequent itemset.
- Let C_k and L_k be the set of candidate k -itemsets and frequent k -itemsets respectively.

The problem of association rule mining can be decomposed into two subproblems [AS94]:

- (i) Find all frequent itemsets;
- (ii) Generate strong association rules from the frequent itemsets.

Note that the second step is straightforward once the first step is done. The overall performance of mining association rules is determined by the first step, which is more computationally expensive and less straightforward [HK01]. Hence, researchers usually tackle the first subproblem only and concentrate on developing efficient algorithms to mine frequent itemsets.

2.3.2 Apriori Algorithm

The Apriori algorithm, proposed by Agrawal and Srikant in 1994 [AS94], is the first successful algorithm for mining association rules, or more precisely, for mining frequent itemsets.

Apriori algorithm, as indicated by its name, uses prior knowledge of frequent itemsets, as we will see below. It employs an iterative level-wise approach to generate frequent $(k+1)$ -itemsets from frequent k -itemsets. It first finds the set of frequent 1-itemsets L_1 and then uses L_1 to generate frequent 2-itemsets L_2 , which in turn is used to generate L_3 . Therefore, in the k^{th} round, frequent k -itemsets are discovered using the frequent $(k-1)$ -itemsets obtained in the $(k-1)^{th}$ round. The process repeats until no more frequent k -itemsets can be found. It requires a full scan of the transaction database for support counting when generating each L_k . In order to reduce the overhead of level-wise generation, an important downward closure property of frequent itemsets, known as *Apriori* property, is used. It states that *all nonempty subsets of a frequent itemset must also be frequent*. Therefore, after generating candidate frequent k -itemsets C_k , one can delete those candidates that contains any $(k-1)$ -subset that is not in L_{k-1} . In this way, Apriori generates much less candidate itemsets than a trivial approach of considering every possible combination of itemsets.

The complete Apriori algorithm is shown in Algorithm 2.1. Algorithm 2.1 also gives the functions used by Apriori. The *Apriori_Gen* function performs two kinds of operations, *join* and *prune*. The join operation combines L_{k-1} with L_{k-1} to generate candidate frequent k -itemsets. The prune operation employs the Apriori property to remove candidates that have an infrequent subset. The test for infrequent subsets is shown in the *Has_Infrequent_Subset* function.

2.3.3 Breaking the Apriori Bottleneck

One problem of the Apriori algorithm is the expensive computational cost due to the generation of candidate itemsets and the multiple scans on the transaction database. That is, it generates C_k from $L_{k-1} \bowtie L_{k-1}$ and each candidate needs to be checked against every transaction in the database. Although the Apriori property helps to reduce the amount of candidates under consideration, it remains to be a bottleneck. Many subsequent works have been done on improving the efficiency of the original algorithm. Several of them are enumerated below.

Algorithm 2.1 Apriori Algorithm for Finding Frequent Itemsets

Input: $\mathcal{D}$, transaction database
 min_sup , minimum support threshold

Output: $\mathcal{L}_f$, all frequent itemsets of length $1 \dots k$

Steps:

- 1: $L_1 = \text{find_frequent_1-itemsets}(\mathcal{D})$;
- 2: **for** ($k = 2$; $L_{k-1} \neq \emptyset$; $k++$) **do**
- 3: $C_k = \text{Apriori-Gen}(L_{k-1}, min_sup)$; *//generate new candidates from $L_{k-1} \bowtie L_{k-1}$*
- 4: **for** each transaction $t \in \mathcal{D}$ **do**
- 5: $C_t = \text{Subset}(C_k, t)$; *//get all candidates contained in t*
- 6: **for** each candidate $c \in C_t$ **do**
- 7: $\text{Support}(c) = \text{Support}(c) + 1$; *//increment the support of c by 1*
- 8: **end for**
- 9: $L_k = \{c \in C_t \mid \text{Support}(c) \geq min_sup \times |\mathcal{D}|\}$
- 10: **end for**
- 11: $\mathcal{L}_f = \bigcup_k L_k$;
- 12: **end for**
- 13: **return** $\mathcal{L}_f$

Function $\text{Apriori_Gen}(L_{k-1}, min_sup)$

- 1: **for** each itemset $l_1 \in L_{k-1}$ **do**
- 2: **for** each itemset $l_2 \in L_{k-1}$ **do**
- 3: **if** $(l_1[1] = l_2[1]) \wedge \dots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ **then** *//join step: generate candidates*
- 4: $c = l_1 \bowtie l_2$;
- 5: **if** $\text{Has_Infrequent_Subset}(c, L_{k-1})$ **then**
- 6: **delete** c ; *//prune step: remove unfruitful candidate*
- 7: **else**
- 8: **add** c **to** C_k ;
- 9: **end if**
- 10: **end if**
- 11: **end for**
- 12: **end for**
- 13: **return** C_k ;

Function $\text{Has_Infrequent_Subset}(c, L_{k-1})$

- 1: **for** each $(k-1)$ -subset s of c **do**
- 2: **if** $s \notin L_{k-1}$ **then**
- 3: **return** TRUE;
- 4: **end if**
- 5: **end for**
- 6: **return** FALSE;

Dynamic Hashing and Pruning (DHP)

After Apriori, the *Dynamic Hashing and Pruning* (DHP) algorithm [PCY95, PCY97] is one of the widely used algorithm for efficient frequent itemset mining. DHP employs a hash technique to reduce the size of candidate itemsets. That is, DHP creates a hash-table in the $(k - 1)^{th}$ pass for the $(k - 1)$ -itemsets in all transactions. Each hash entry contains the $(k - 1)$ -itemsets sharing the same value and a count of such itemsets. An k -itemset is eligible as a candidate only if it is hashed into an entry whose count is no less than the minimum support count required, or discarded otherwise. In addition to reducing the number of candidate itemsets, DHP also uses some pruning strategies to reduce the size of the transaction database in each iteration. Firstly, items that do not appear in any of the frequent itemsets are removed from each transaction. Secondly, a transaction is removed from the database if the transaction does not contain any frequent itemset. DHP has significant speed improvements due to the reduced size of the generated candidate itemsets. However, it incurs additional overhead because of the need to do hashing and maintain the hash table. The experimental results have revealed that the hash technique should only be applied to the generation of candidate 2-itemsets, leading to speed improvement of up to three times faster than Apriori.

Partitioning

In order to reduce the I/O cost of multiple database scans, Savasere *et al.* proposed a two-pass partitioning algorithm [SON95], which requires just two database scans to mine the frequent itemsets by keeping the database in memory. The main idea of this algorithm is the use of a portion of the main memory for storing the database partitions. The number of partitions in the database is determined based on the size of the available memory buffer. Therefore, the database is divided into multiple partitions that do not overlap with one another. When each partition is read, it fits properly into the memory buffer. Given that there are n partitions, then the minimum support threshold for each partition will be $\frac{min_sup}{n}$. For each partition, all frequent itemsets within the partition are found, which are referred to as *local frequent itemsets*. All of the local frequent k -itemsets ($k = 1, 2, \dots$) can be found in just one scan on the database by employing a vertical data structure that, for each itemset, records the TIDs of the transactions containing this itemset. Note that all local frequent itemsets are combined to form the candidate itemsets, called *global candidate*

itemsets, with respect to the entire database. In the second scan, the actual support of each global candidate itemset is counted against the *min_sup* threshold in order to generate the *global frequent itemsets*, i.e., the complete set of frequent itemsets with respect to the entire database.

One key observation about the partitioning algorithm is the need to effectively count the candidate 2-itemsets. As discussed in [PCY95, PCY97], the largest number of candidates are those of size two². In [AS94], the authors also reported that the AprioriTid algorithm, which is a variation of Apriori using TID intersections, was inefficient during the early passes. The reason lies in the fact that the intersection of TIDs for all these candidates are more expensive than the Apriori's horizontal scan of the database. Hence, the performance of the partitioning algorithm may be improved by considering the use of horizontal database scans in the second pass.

Novel Data Structures

In [Zak00b], Zaki *et al.* presented new algorithms based on *lattice theory* [DP90] in an attempt to decompose the large frequent itemset search space into smaller segments that can be handled separately in memory. A vertical database format was adopted to facilitate the use of lattices, where each itemset is associated with a list of transactions, known as *TID-list*, that contain this itemset. This can significantly speed up the support counting of itemsets. However, this approach causes extra overhead because most transaction databases are in the horizontal format and thus need to be transformed into vertical format before mining. In [Zak00b], six new algorithms are proposed, among which the best one *MaxClique* performs about an order of magnitude faster than Apriori at support thresholds of 0.75% and lower. Recently, Zaki *et al.* improved upon their previous algorithms by introducing a new vertical data representation known as *Diffset* [ZG01]. As diffsets store only differences in the TIDs for a candidate itemset, they require far less memory and are hence more scalable and efficient. Together with a proposed algorithm *dEclat*, diffsets have further pushed the speed limit to a few orders of magnitude faster than Apriori.

Approximately six years after the introduction of Apriori algorithm, Han *et al.* [HP00, HPY00] broke the candidate generation bottleneck and achieved a breakthrough in the

²This statement may not be true in some circumstances, especially with the appearance of long frequent itemsets. Consider an extreme example, every transaction contains the same 100 items $i_1, i_2, \dots, i_{100}$. In this case, $|C_2| = |L_2| = \binom{100}{2} = 4950$ and $C_3 = L_3 = \binom{100}{3} = 161700$. It is clear that C_2 is not the largest set.

algorithmic aspects of mining frequent itemsets by using a novel data structure known as *Frequent Pattern-tree* (FP-tree). FP-tree is a compressed representation of the database storing information about frequent 1-itemsets, i.e., frequent items. It can be constructed using only 2 database scans. During the first scan, frequent items L_1 are obtained and sorted in support count descending order. In the second scan, frequent items in each transaction are first sorted based on the order of L_1 and then inserted into the FP-tree. The infrequent items are removed from the database at the same time. During the insertion process, a new branch will be created for each transaction and different transactions having common frequent items may share a common *prefix* path.

Instead of the level-wise candidates generation and examination, the process of finding frequent itemsets is converted to a recursive mining process called *FP-growth*. This technique first constructs a sub-tree by examining branches in the FP-tree containing the suffix itemset of interest. This sub-tree is called the *conditional FP-tree*, and is then subsequently used to enumerate all frequent itemsets of different sizes containing the same suffix. Unlike candidate generation, the focus of FP-growth is in support count accumulation and adjustment. This is computationally cheaper as compared to candidate generation and test. In addition, the FP-tree is compact enough to reside in the main memory. This gives a dramatic improvement in speed of mining. As reported by Han *et al.*, FP-growth could be an order of magnitude faster than Apriori for small *min_sup* threshold.

FP-growth uses a partition-based, *divide-and-conquer* approach as compared to the Apriori's bottom-up candidate generation. This reduces the size of the conditional FP-tree and inherently transforms the problem of finding long itemsets to looking for shorter ones and then concatenating with the suffix. Instead of illustrating the details of FP-growth, we refer readers to [HP00, HPY00, HPYM04] where the authors have given examples to illustrate the technique.

However, FP-growth only achieves significant speed-ups with the appearance of long itemsets and large portion of overlapping between transactions. Moreover, it is only incremental to a certain extent depending on the *min_sup* threshold. That is, FP-tree must be reconstructed for a smaller *min_sup*.

2.4 Time Series Analysis

Group pattern mining deals with time series of user location data. Hence, it is related to the previous works on time series analysis. A time series database consists of sequences

of values recorded at regular time intervals. Time series databases are popular in many applications, such as the daily fluctuations of exchange rates. In the following, we give a brief overview of some time series analysis research.

- *Mining cyclic association rules.* In [ORS98], B. Ozden *et al.* introduced the problem of finding cyclic association rules from transaction databases. An association rule is referred to as cyclic if it holds at regular time intervals. That is, a cyclic association rule should satisfy the minimum support and minimum confidence thresholds based on transactions occurred during periodic time intervals, instead of the whole transaction database. Cyclic association rules fall into the category of *full periodic patterns*, which have been studied in signal processing, as cyclic association rules must hold in *every* periodic time interval. Full periodic patterns refers to the patterns in which every time point precisely or approximately contributes to a cyclic behavior [HDY99], which is less common in practice than *partial periodic patterns* discussed below.
- *Mining partial periodic patterns.* Partial periodic patterns are defined as patterns that specify the periodic behavior of the time series at some but not all of the time points [HGY98, HDY99]. For example, Frank goes to office from 8:30 to 9:00 every weekday morning, but his activities at other times do not have much regularity. It is clear that partial periodicity is more common in practice since it is likely that only some not all of the time points appear periodically. In [HDY99], Han *et al.* found that the Apriori property still holds for partial periodic patterns and proposed Apriori-like algorithms to mine single period patterns. In addition, they further designed another FP-tree-like structure known as max-subpattern tree to improve mining efficiency.

A lot of studies on mining periodic patterns from time series have been reported by J. Yang *et al.* [YWY00, WYY00, WYY01a, WYY01b, WYY02, YWY02, YYWH02]. There are mainly four novel methods proposed in their work, including *asynchronous pattern* method [YWY00, WYY00, YWY02], *meta-pattern* method [WYY01b], *information gain* method [WYY01a, WYY02] and *compatibility-matrix* method [YYWH02]. Asynchronous pattern is designed to handle the cases that the presence of periodic patterns may be misaligned due to insertion of random noise. That is, instead of computing support by exact pattern matching, asynchronous pattern allows partial pattern matching to give more flexibility. Meta-pattern further extends asynchronous

pattern in the sense that it allows a pattern at high levels *contain* some other patterns at low levels as components, in order to reveal more patterns at higher levels even with the presence of noise. Information gain is introduced in [WYY01a] as a new interestingness measure. Different from support measure, where each occurrence of a pattern equally contributes to the interestingness, information gain measures the degree of interestingness of each occurrence of a pattern as a function of its probability of occurrence. This may be useful in applications such as computational biology. In [YYWH02], the authors considered the situation where the element mutation is allowed, such as the cases in protein sequences, and thus proposed a compatibility matrix to build probabilistic connections between the observed element and the actual element in order to remove the impact of mutation.

However, the above work on mining periodic patterns does not address the problem of deriving a suitable period or a range of possible periods before mining, which is the main concern in [MH01]. Ma *et al.* considered the problem of discovering partially periodic temporal associations, known as *p-patterns*, with unknown periods. They developed an algorithm for finding periods based on a chi-squared test and studied the performance of this algorithm in the presence of noise. Two algorithms, *association-first* algorithm and *period-first* algorithm, for discovering p-patterns have been developed, based on whether periods or temporal associations are discovered first. The experiment results have shown that the association-first algorithm has a higher tolerance to noise while the period-first algorithm is more efficient.

Note that the periodic analysis and group pattern mining deal with different mining tasks. Firstly, the former performs mining on time series database, which usually consists of symbolic data and do not involve spatial information. On the other hand, group pattern mining deals with time series of locations of mobile users. Secondly, the periodic analysis attempts to find repeated behaviors, either full periodic or partial periodic, of the symbols with time stamps. Such patterns can be viewed as *intra-database* patterns since all symbols are contained in the time series database. On the contrary, group patterns are *inter-database* patterns since the purpose is to find the grouping information among the mobile users, where each user has a time series of his/her locations.

- *Similarity search in time series.* Vlachos *et al.* presented techniques to compute the similarities between trajectories, i.e., time series of locations, of moving objects in [VKG02]. The trajectory similarity is defined by extending the *longest common subsequence* (LCSS) model [AFS93, ALSS95] to consider the shift in space when comparing two trajectories. Note that, if the spatial shift is not allowed, two moving objects are considered similar if their trajectories are close to each other. Nevertheless, this work focused on designing efficient algorithm to find a trajectory, from a set of trajectories, that is most similar to a given *query trajectory*. Temporal constraints, such as minimum duration threshold in group pattern mining, are not taken into consideration.

2.5 Sequential Pattern Mining

Note that, a time-series database is also a sequence database. However, the reverse does not always hold. A sequence database can be any database containing sequences of ordered events, with or without concrete notions of time [HK01].

Sequential pattern mining was first introduced by Agrawal and Srikant in [AS95] and was generalized by the same authors in [SA96]. Given a sequence database, in which each sequence is an ordered list of transactions that contains a set of items and has a time stamp, the goal of sequential pattern mining is to find the set of subsequences with support exceeding a minimum support threshold, where the support of a subsequence is the number of sequences containing this subsequence.

Most of the previous research efforts have been put into developing efficient sequential pattern mining algorithms. An Apriori-like algorithm GSP was first proposed in [SA96], based on Apriori property which indicates that if a sequential pattern of length k is infrequent, its supersequence of length $k + 1$ can not be frequent either. GSP needs to generate and examine a huge number of intermediate candidate subsequences. To break this bottleneck, J. Han *et al.* developed an efficient FP-growth-like algorithm PrefixSpan for mining large sequence database containing long sequences [PHA⁺01, PHW02, PHA⁺04]. The PrefixSpan algorithm greatly reduces the overhead of candidate subsequence generation by transforming the sequence database into a set of smaller projected database and performing mining on each projected database. Another efficient sequential pattern mining

algorithm, known as SPADE, was proposed by Zaki in [Zak01], in which the author adopted vertical format data and decomposed the original problem into smaller subproblems.

If applying sequential pattern mining algorithms to a movement database containing sequences of locations for each mobile user. The mining results will be a set of ordered location subsequences frequently visited by mobile users. Such results are different from that in group pattern mining. Besides, the sequence database only contains *nominal* time stamps. That is, the time stamps are in the form of index number to order the transactions that occur at different time, instead of indicating the actual transaction time. Hence, the interval of two time stamps carries no information with respect to the real time interval. In contrast, group pattern mining concerns about the real time intervals during which group members maintain close distances.

2.6 Mining Spatial or Spatio-Temporal Data

As we mentioned above, group pattern mining deals with time series of user location information involving both temporal and spatial dimensions.

There has been much work on spatial data mining, such as spatial relationship or spatial trend mining to find relationships between spatial objects or detect changes or trends along a spatial dimension [EKS97, EFKS98, EFKS00], spatial association rule mining to find association rules of spatial objects [KH95], spatial cluster analysis to identify clusters or densely populated regions according to some distance measure [NH94, EKX95, EKXS96, SKKL98], co-location patterns to discover spatial objects that often co-located together [SH01], etc.. However, most of these work only focuses on space dimension, without consideration of time. For example, in [SH01], S. Shekhar *et al.* proposed approaches to discover spatial co-location patterns, which is defined as a set of spatial features that are physically close to one another. However, the locations of spatial features are *static*, which is not the case in group pattern mining.

In addition, some researchers have paid attention to spatio-temporal mining by taking both temporal and spatial aspects into consideration [Sel99, RHS00, RL01]. However, in this relatively new area, the existing work is mainly focused on the models and structures for *indexing* the moving objects [PSZ99, FGNS00], without referring to what kind of interesting knowledge can be derived from the spatio-temporal database and how to find it. On the other hand, group pattern mining is different in the sense that the concept of moving objects

are confined as mobile users, and a novel group pattern model is designed to characterize some interesting grouping knowledge.

2.7 Groups in Mobile Computing

In mobile computing, in order to reduce the uplink bandwidth consumption during location updating especially for a large population, several group models have been proposed such that location updating is conducted only for each derived group rather than for each individual user or node [RHH01, HG02, HK02, LLC04]. However, the groups defined and discovered in these works are derived only based on physical closeness, without considering time information. In addition, there is no interestingness measure for the derived groups, as there is no need to rank the groups in the location update problem. The work is concerned with groups of nodes for efficient location updating, not people with relationships.

Chapter 3

Group Pattern Mining

3.1 Problem Definition

Group pattern mining is to be conducted on a user movement database defined by $D = (D_1, D_2, \dots, D_M)$, where D_i is a time series of tuples $(t, (x, y, z))$ denoting the x -, y - and z -coordinates of user u_i at time t . We assume that there are N time points in the time series, and therefore t is an integer ranging from 0 to $N-1$. For simplicity, we denote the location of a user u_i at time t by $u_i[t].p$, and his/her x -, y -, and z - values at time t by $u_i[t].x$, $u_i[t].y$ and $u_i[t].z$ respectively. A very small user movement database example is shown in Table 3.1.

Definition 3.1 *Given a set of users G , a maximum distance threshold max_dis , and a minimum time duration threshold min_dur , a set of consecutive time points $[t_a, t_b]$ is called a **valid segment** of G , if*

- (i) $\forall u_i, u_j \in G, \forall t, t_a \leq t \leq t_b, d(u_i[t].p, u_j[t].p) \leq max_dis$;
- (ii) $t_a = 0$ or $\exists u_i, u_j \in G, d(u_i[t_a - 1].p, u_j[t_a - 1].p) > max_dis$;
- (iii) $t_b = N - 1$ or $\exists u_i, u_j \in G, d(u_i[t_b + 1].p, u_j[t_b + 1].p) > max_dis$;
- (iv) $(t_b - t_a + 1) \geq min_dur$.

In other words, within a valid segment of a set of users G , all members must be close to one another for at least a minimum time duration (min_dur). The distance function, $d()$, returns the distance between two points¹. Furthermore, valid segments are maximal as no

¹In this research, Euclidean distance is adopted. However, other distance metrics can be used for different applications, such as Manhattan distance and *weighted* Euclidean distance, as long as they satisfy the following four properties: (1) $d(i, j) \geq 0$; (2) $d(i, i) = 0$; (3) $d(i, j) = d(j, i)$; and (4) $d(i, j) \leq d(i, h) + d(h, j)$.

Table 3.1: User Movement Database D

u_1				u_2				u_3			
t	x	y	z	t	x	y	z	t	x	y	z
0	154.56	21.12	14.87	0	156.18	27.75	16.25	0	162.68	26.5	13.53
1	154.56	19.25	14.87	1	162	26.5	16.25	1	162.68	26.5	13.53
2	158.12	21.12	14.87	2	157.93	21.06	16.25	2	162.68	26.5	13.53
3	157.68	26.5	14.87	3	162.56	26.68	16.25	3	161.12	23.06	13.53
4	157.68	26.5	14.87	4	162.56	26.68	16.25	4	157.62	24.5	13.53
5	157.68	26.5	14.87	5	166.68	23.25	16.25	5	159.31	19.31	13.53
6	157.68	26.5	14.87	6	162	28.56	16.25	6	159.12	23.56	13.53
7	152.75	26.87	14.87	7	163.56	25.81	16.25	7	159.43	24.93	13.53
8	152.75	26.87	14.87	8	158.31	24.93	16.25	8	155.68	26.75	12.18
9	152.75	26.87	14.87	9	158.31	24.93	16.25	9	149.56	21.87	12.18

u_4				u_5				u_6			
t	x	y	z	t	x	y	z	t	x	y	z
0	158.68	20.43	16.25	0	159.56	26.06	16.25	0	162.31	14.87	16.25
1	154.75	25.25	17.59	1	159.56	26.06	16.25	1	163.06	17.87	16.25
2	158.06	27.93	17.59	2	159.56	26.06	16.25	2	161.5	15.81	16.25
3	158.06	24.25	17.59	3	159.56	26.06	16.25	3	156.87	21.18	17.59
4	158.06	24.25	17.59	4	163.37	27.75	16.25	4	161.62	25.43	17.59
5	156.93	23.37	17.59	5	163.37	27.75	16.25	5	160.87	25.37	17.59
6	161.62	20.62	17.59	6	168.12	25.37	16.25	6	166.12	28.37	17.59
7	167.68	21.37	17.59	7	164.75	28.12	16.25	7	160.06	25.81	17.59
8	170.25	22.56	17.59	8	164.75	28.12	16.25	8	163.62	27.12	17.59
9	165.06	25.81	17.59	9	164.75	28.12	16.25	9	165.18	27.12	17.59

two valid segments of the same set of users can overlap each other. The thresholds, max_dis and min_dur , are used to define the spatial and temporal proximity requirements between members of a group. In particular, the min_dur threshold helps to weed out short-lived or accidental closeness between users. Let s be a valid segment, $s = [t_a, t_b]$. The *length* of s , $|s|$, is defined as the number of time points contained in s , i.e., $|s| = t_b - t_a + 1$.

Consider the user movement database in Table 3.1. Suppose $min_dur = 3$ and $max_dis = 10$. The distances between u_1 and u_2 are: $d(u_1[0].p, u_2[0].p) = 6.96317$, $d(u_1[1].p, u_2[1].p) = 10.4795$, $d(u_1[2].p, u_2[2].p) = 1.39431$, $d(u_1[3].p, u_2[3].p) = 5.07456$, $d(u_1[4].p, u_2[4].p) = 5.07456$, $d(u_1[5].p, u_2[5].p) = 9.66783$, $d(u_1[6].p, u_2[6].p) = 4.981$, $d(u_1[7].p, u_2[7].p) = 10.9492$, $d(u_1[8].p, u_2[8].p) = 6.04827$, $d(u_1[9].p, u_2[9].p) = 6.04827$. Since u_1 and u_2 are close in $[2, 6]$, which has more than 3 time points, $[2, 6]$ is a valid segment of $\{u_1, u_2\}$. Ta-

ble 3.2 shows all the closeness relationships between the six users throughout the ten time points, where the closeness of each user pair is assigned a “+” when they are not more than max_dis apart, or a “−” otherwise.

Table 3.2: Closeness between Users in Table 3.1 ($max_dis = 10$)

t	u_1					u_2				u_3			u_4		u_5
	u_2	u_3	u_4	u_5	u_6	u_3	u_4	u_5	u_6	u_4	u_5	u_6	u_5	u_6	u_6
0	−	−	+	+	−	+	+	+	−	+	+	−	+	−	−
1	−	−	+	+	+	+	+	+	+	+	+	−	+	−	−
2	+	+	+	+	+	+	+	+	+	+	+	−	+	−	−
3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
4	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
5	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
6	+	+	+	−	+	+	+	+	+	+	+	+	+	+	+
7	−	+	−	−	+	+	+	+	+	+	+	+	+	+	+
8	−	+	−	−	−	+	−	+	+	−	−	+	+	+	+
9	−	+	−	−	−	−	−	+	+	−	−	−	+	+	+

Definition 3.2 Given a set of users G , thresholds max_dis and min_dur , we say that G , max_dis and min_dur form a **group pattern**, denoted by $P = \langle G, max_dis, min_dur \rangle$, if G has a valid segment.

The valid segments of a group pattern P are those of its G component. We also call a group pattern with k users a **k -group pattern**.

Definition 3.3 Given two group patterns, $P = \langle G, max_dis, min_dur \rangle$ and $P' = \langle G', max_dis, min_dur \rangle$, P' is called a **sub-group pattern** of P , if $G' \subseteq G$.

In a movement database, a group pattern may have multiple valid segments. The combined length of these valid segments is called the *weight count* of the pattern. We therefore quantify the significance of the pattern by comparing its weight count with the overall log duration.

Definition 3.4 Let P be a group pattern with valid segments $s_1, \dots, s_n$, the **weight count** and **weight** of P are defined as:

$$\text{weight count}(P) = \sum_{i=1}^n |s_i| \quad (\text{Eq. 3.1})$$

$$\text{weight}(P) = \frac{\sum_{i=1}^n |s_i|}{N} \quad (\text{Eq. 3.2})$$

Since weight represents the *proportion* of the time points a group of users stay close together, the larger the weight is, the more significant (or interesting) the group pattern is. A group pattern is called a **valid group pattern** if its weight exceeds a threshold min_wei , and the corresponding group of users is called a **valid group**. For example, suppose $\text{min_wei} = 50\%$. The group pattern $\langle \{u_1, u_2\}, 10, 3 \rangle$ is a valid group pattern, since it has a valid segment $\{[2, 6]\}$ and a weight of $\frac{5}{10} \geq 0.5$, and $\{u_1, u_2\}$ is a valid 2-group.

Given a valid group G , we use $s(G)$ to denote the set of valid segments of G , e.g., $s(\{u_1, u_2\}) = \{[2, 6]\}$. In addition, the complete set of valid groups is denoted by $\mathbb{G}$ and we use $s(\mathbb{G})$ to denote the set of valid segment lists for all valid groups in $\mathbb{G}$, i.e., $s(\mathbb{G}) = \{s(G) \mid G \in \mathbb{G}\}$.

Throughout this thesis, we use *valid group pattern* and *valid group* interchangeably since the former is just the latter combined with max_dis and min_dur thresholds.

Definition 3.5 Given the thresholds max_dis , min_dur , and min_wei , the problem of finding the set of all valid groups is known as **valid group mining**.

We address the valid group mining problem in the rest of this chapter and the next chapter.

Discussions

There are some similarities between group pattern mining and the classical association rule mining. In the latter problem, the goal is to discover all frequent itemsets, which is defined as a set of items, with support exceeding a minimal support threshold. There are however several key differences that render the direct application of association rule mining methods not feasible in valid group pattern mining.

- There is no explicit concept of *transaction* in a movement database. The movement database consists of multiple time series of locations, one for each user. One can try to organize the locations recorded at one time point into some kind of transactions with each transaction representing a set of locations that are not more than *max_dis* apart at that time point. For example, 8 transactions can be derived based on the locations at time point 9 in Table 3.1: $\{u_1, u_3\}$, $\{u_2, u_5\}$, $\{u_2, u_6\}$, $\{u_4, u_5\}$, $\{u_4, u_6\}$, $\{u_5, u_6\}$, $\{u_2, u_5, u_6\}$, and $\{u_4, u_5, u_6\}$. However, this grouping of user locations at each time point into transactions can lead to extremely large number of transactions, especially for a large population. Such a database transformation overhead can be prohibitive if there are many time points and users.
- The *weight* defined for valid group pattern mining is very different from the *support* defined in association rule mining. By transactionizing the movement database, it does not address the weight counting problem. For transactions derived from a single time point, we should not double count the transactions that have the same set of users. In the above example, user pair $\{u_2, u_5\}$ is contained in two derived transactions $\{u_2, u_5\}$ and $\{u_2, u_5, u_6\}$. But the weight count of $\{u_2, u_5\}$ should be only incremented by 1, instead of 2, since both two transactions occur at the same time point 9.

Therefore, it is necessary to design new algorithms for mining valid group patterns.

3.2 Mining Valid Groups

In [WLH03], we proposed two algorithms to mine the exact set of valid groups, known as the *Apriori-like Group Pattern mining (AGP)* algorithm and *Valid Group Growth (VG-growth)* algorithm. The former explores the Apriori property of valid group patterns and extends the Apriori algorithm [AS94] to mine valid groups. The latter is based on idea similar to the FP-growth algorithm [HPY00]. In the following, we present the proposed AGP and VG-growth algorithms.

3.2.1 AGP Algorithm

AGP algorithm is built upon the Apriori property that also holds for group patterns.

Theorem 3.1 [*Apriori property of group patterns*]: *If a group pattern is valid, then all of its sub-group patterns are valid as well.*

Proof. Given min_wei , and a group pattern $P = \langle G, min_dur, max_dis \rangle$, if P is a valid group pattern, then $\frac{\sum |s_i|}{N} \geq min_wei$, where s_i 's are valid segments of P . Let P' denote any sub-group pattern of P . Note that, for each valid segment s_i of P , there must exist a valid segment s'_i of P' such that $s_i \subseteq s'_i$ and so $|s'_i| \geq |s_i|$. Hence, we have $\frac{\sum |s'_i|}{N} \geq \frac{\sum |s_i|}{N} \geq min_wei$. That is, the sub-group pattern P' is also a valid group pattern. ■

The AGP algorithm is shown in Algorithm 3.1. We use C_k to denote the set of candidate k -groups, and $\mathbb{G}_k$ to denote the set of valid k -groups. The AGP algorithm starts by mining $\mathbb{G}_1$, the set of all distinct users. It then uses $\mathbb{G}_1$ to find $\mathbb{G}_2$, which is in turn used to find $\mathbb{G}_3$. The process repeats until no more valid k -groups can be found. In each iteration, Apriori property is used to generate candidate groups of larger size, and to prune the unpromising candidate groups. However, there are two key differences between AGP and the standard Apriori algorithm:

Algorithm 3.1 AGP Algorithm.

Input: D, max_dis, min_dur , and min_wei
Output: $\mathbb{G}$
Steps:
1: $\mathbb{G} = \emptyset$;
2: $\mathbb{G}_1 =$ the set of all distinct users;
3: **for** ($k = 2$; $\mathbb{G}_{k-1} \neq \emptyset$; $k++$) **do**
4: $C_k = \text{Generate_Candidate_Groups}(\mathbb{G}_{k-1})$;
5: **for** ($t = 0$; $t < N$; $t++$) **do**
6: **for** each candidate k -group $c_k \in C_k$ **do**
7: **if** $\text{Is_Close}(c_k, t, max_dis)$ **then**
8: $c_k.cur_seg++$;
9: **else**
10: **if** $c_k.cur_seg \geq min_dur$ **then**
11: $c_k.weight++ = c_k.cur_seg$;
12: **end if**
13: $c_k.cur_seg = 0$;
14: **end if**
15: **end for**
16: **end for**
17: $\mathbb{G}_k = \{c_k \in C_k \mid c_k.weight \geq min_wei \cdot N\}$;
18: $\mathbb{G} = \mathbb{G} \cup \mathbb{G}_k$;
19: **end for**
20: **return** $\mathbb{G}$;

Algorithm 3.2 Functions used by AGP Algorithm.**Function Generate_Candidate_Groups**($\mathbb{G}_{k-1}$)

```

1: for each  $(k-1)$ -group  $g_{k-1} \in \mathbb{G}_{k-1}$  do  $//g_{k-1} = (u_1, \dots, u_{k-1})$ 
2:   for each  $(k-1)$ -group  $h_{k-1} \in \mathbb{G}_{k-1}$  do  $//h_{k-1} = (v_1, \dots, v_{k-1})$ 
3:     if  $(u_1 = v_1) \wedge \dots \wedge (u_{k-2} = v_{k-2}) \wedge (u_{k-1} < v_{k-1})$  then
4:        $c_k = (u_1, \dots, u_{k-2}, u_{k-1}, v_{k-1})$ ;  $//join\ step$ 
5:       if Has_Invalid_Subgroup( $c_k, \mathbb{G}_{k-1}$ ) then
6:         delete  $c_k$ ;  $//prune\ step$ 
7:       else
8:          $c_k.cur\_seg = 0$ ;
9:          $c_k.weight = 0$ ;
10:        add  $c_k$  to  $C_k$ ;
11:      end if
12:    end if
13:  end for
14: end for
15: return  $C_k$ ;

```

Function Has_Invalid_Subgroup($c_k, \mathbb{G}_{k-1}$)

```

1: for each  $(k-1)$ -subgroup  $c_{k-1}$  of  $c_k$  do
2:   if  $c_{k-1} \notin \mathbb{G}_{k-1}$  then
3:     return TRUE;
4:   end if
5: end for
6: return FALSE;

```

Function Is_Close(c_k, t, max_dis)

```

1: for each pair of users  $(u_i, u_j)$  in  $c_k$  do
2:   if  $d(u_i[t].p, u_j[t].p) > max\_dis$  then
3:     return FALSE;
4:   end if
5: end for
6: return TRUE;

```

- (i) Instead of examining whether a transaction contains a candidate itemset, AGP tests whether users in a candidate group are *close* to one another at a given time point;
- (ii) Instead of simply incrementing support counts, AGP accumulates the lengths of all valid segments so as to compute the weight count of a candidate group.

For example, suppose we want to mine valid groups from the movement database D shown in Table 3.1 with $max_dis = 10$, $min_dur = 3$, and $min_wei = 50\%$. AGP finds all valid groups as follows.

- $\mathbb{G}_1$ is first assigned as the set $\{ \{u_1\}, \{u_2\}, \{u_3\}, \{u_4\}, \{u_5\}, \{u_6\} \}$.
- AGP then generates C_2 by a *join* operation, which is the same as that in Apriori algorithm (see Algorithm 2.1). $C_2 = \{ \{u_1, u_2\}, \{u_1, u_3\}, \{u_1, u_4\}, \{u_1, u_5\}, \{u_1, u_6\}, \{u_2, u_3\}, \{u_2, u_4\}, \{u_2, u_5\}, \{u_2, u_6\}, \{u_3, u_4\}, \{u_3, u_5\}, \{u_3, u_6\}, \{u_4, u_5\}, \{u_4, u_6\}, \{u_5, u_6\} \}$.
- Then AGP scans D to compute the weight counts for each candidate 2-group and selects the valid ones. In this example, all candidate 2-groups are valid, i.e., $\mathbb{G}_2 = C_2$.
- From $\mathbb{G}_2$, AGP generates C_3 . $C_3 = \{ \{u_1, u_2, u_3\}, \{u_1, u_2, u_4\}, \{u_1, u_2, u_5\}, \{u_1, u_2, u_6\}, \{u_1, u_3, u_4\}, \{u_1, u_3, u_5\}, \{u_1, u_3, u_6\}, \{u_1, u_4, u_5\}, \{u_1, u_4, u_6\}, \{u_1, u_5, u_6\}, \{u_2, u_3, u_4\}, \{u_2, u_3, u_5\}, \{u_2, u_3, u_6\}, \{u_2, u_4, u_5\}, \{u_2, u_4, u_6\}, \{u_2, u_5, u_6\}, \{u_3, u_4, u_5\}, \{u_3, u_4, u_6\}, \{u_3, u_5, u_6\}, \{u_4, u_5, u_6\} \}$.
- AGP scans D again and removes four invalid 3-groups: $\{u_1, u_2, u_5\}$, $\{u_1, u_3, u_5\}$, $\{u_1, u_4, u_6\}$, and $\{u_1, u_5, u_6\}$. Thus, $\mathbb{G}_3 = \{ \{u_1, u_2, u_3\}, \{u_1, u_2, u_4\}, \{u_1, u_2, u_6\}, \{u_1, u_3, u_4\}, \{u_1, u_3, u_6\}, \{u_1, u_4, u_5\}, \{u_2, u_3, u_4\}, \{u_2, u_3, u_5\}, \{u_2, u_3, u_6\}, \{u_2, u_4, u_5\}, \{u_2, u_4, u_6\}, \{u_2, u_5, u_6\}, \{u_3, u_4, u_5\}, \{u_3, u_4, u_6\}, \{u_3, u_5, u_6\}, \{u_4, u_5, u_6\} \}$.
- From $\mathbb{G}_3$, AGP generates C_4 . $C_4 = \{ \{u_1, u_2, u_3, u_4\}, \{u_1, u_2, u_3, u_6\}, \{u_1, u_2, u_4, u_6\}, \{u_1, u_3, u_4, u_6\}, \{u_2, u_3, u_4, u_5\}, \{u_2, u_3, u_4, u_6\}, \{u_2, u_3, u_5, u_6\}, \{u_2, u_4, u_5, u_6\}, \{u_3, u_4, u_5, u_6\} \}$. Note that, $\{u_1, u_2, u_4, u_6\}$ and $\{u_1, u_3, u_4, u_6\}$ are subsequently pruned from C_4 based on Theorem 3.1 since both of them have an invalid sub-group $\{u_1, u_4, u_6\}$ which is not in $\mathbb{G}_3$.
- AGP then scans D again to obtain $\mathbb{G}_4$ as $\{ \{u_1, u_2, u_3, u_4\}, \{u_2, u_3, u_4, u_5\}, \{u_2, u_3, u_4, u_6\}, \{u_2, u_3, u_5, u_6\}, \{u_2, u_4, u_5, u_6\}, \{u_3, u_4, u_5, u_6\} \}$.
- From $\mathbb{G}_4$, AGP generates C_5 , which only contains one candidate 5-group $\{u_2, u_3, u_4, u_5, u_6\}$.
- After scanning D again, AGP generates $\mathbb{G}_5 = \{ \{u_2, u_3, u_4, u_5, u_6\} \}$.
- AGP terminates here and the discovered valid groups are $\mathbb{G}_2 \cup \mathbb{G}_3 \cup \mathbb{G}_4 \cup \mathbb{G}_5$.

Figure 3.1 portrays the search tree for algorithm AGP ². In the search tree, each node with depth ≥ 2 represents a valid group. Given a valid group G , we use $n(G)$ to denote

²In Figure 3.1, we only give the number of user id to save space, e.g., 1, 2 represents $\{u_1, u_2\}$.

the corresponding node in the search tree. It is clear that AGP performs a level-wise breadth-first traversal on the search tree.

Figure 3.1: Search Tree for AGP

Time Complexity Analysis

In function *Generate_Candidate_Groups* (see Algorithm 3.2), each call to function *Has_Invalid_Subgroups* (line 5) requires $O(k \cdot |\mathbb{G}_{k-1}|)$ time in the worst case. With the two loops in Lines 1 and 2, the time complexity of function *Generate_Candidate_Groups* is $O(k \cdot |\mathbb{G}_{k-1}|^3)$.

In Algorithm 3.1, Lines 5 - 16 scan the database to compute the weight, which costs³ $O(M \cdot N, N \cdot |C_k| \cdot \binom{k}{2}) = O(M \cdot N, N \cdot |C_k| \cdot k^2)$, where M is the number of distinct users and N is the whole time span of D . Line 17 selects the valid groups, which costs $O(|C_k|)$.

In total, the time cost of AGP is $O(\Sigma_k \{k \cdot |\mathbb{G}_{k-1}|^3, M \cdot N, N \cdot |C_k| \cdot k^2\})$.

This is a main memory based analysis, which does not consider the disk access overhead. That is, both the movement database and the candidate sets are assumed to reside in main memory. Note that, the $(N \cdot |C_k| \cdot k^2)$ component represents the overhead of scanning D to check the distance between every two users of every candidate group. This is the

³We use $O(A, B, C, \dots)$ to denote $\max\{O(A), O(B), O(C), \dots\}$.

main bottleneck of all Apriori-like algorithms [HP96] and we therefore develop algorithm VG-growth to reduce the overhead.

3.2.2 VG-growth Algorithm

AGP algorithm, inherited from the Apriori algorithm, incurs large overhead in candidate k -group ($k > 2$) generation and database scans to check whether the candidates are valid. In order to reduce such overhead, we further proposed *VG-growth* algorithm in [WLH03], which uses a novel data structure known as *VG-graph*. VG-growth and VG-graph are designed based on the principle similar to that of FP-growth and FP-tree [HPY00] for association rule mining.

In [HPY00], Han *et al.* proposed a novel data structure known as *FP-tree* and a divide-and-conquer algorithm, *FP-growth*, that mines association rules without candidate generation. In a FP-tree (Frequent Pattern tree), each node represents a frequent item, and the frequent items are ordered in support descending order so that the more frequently occurring items are more likely to be shared and thus located closer to the top of the FP-tree. The FP-growth method starts from a frequent item (as an initial *suffix* pattern), examines only its *conditional pattern base* (a “sub-database” which contains the set of frequent items co-occurring with the suffix pattern), constructs its *conditional FP-tree*, and performs mining recursively with such a tree. The major operations of mining are *count accumulation* and *prefix count adjustment*, which are usually much less costly than candidate generation and pattern matching operations performed in the Apriori-like algorithm.

As we extend the FP-tree structure and FP-growth algorithm to valid group pattern mining, the unique features of valid group pattern mining have to be considered, which will make some basic concepts adopted by FP-tree and FP-growth not applicable.

- Each node in a FP-tree is a frequent item, which is also the smallest unit in association rule mining. However, in valid group pattern mining, the smallest unit is a valid 2-group. A direct construction of FP-tree-like structure based on valid 2-groups will lead to excessive number of nodes in the tree.
- The *weight* used in valid group pattern mining is more complicated than the *support* measure. Hence, it is necessary to store the list of valid segments for each valid 2-group so as to derive the weight counts of valid groups of larger sizes.

Construction of VG-graph

Definition 3.6 A *valid group graph (VG-graph)* is a weighted undirected graph (V, E, s) , where

- (i) $V = \{u \mid u \in G, G \in \mathbb{G}_2\}$. That is, each vertex in V represents a user who participates in some valid 2-group. This set of users is also known as valid users.
- (ii) $E = \{(u_i, u_j) \mid \{u_i, u_j\} \in \mathbb{G}_2\}$. That is, each edge in E represents a valid 2-group.
- (iii) s is a function that maps each edge to the corresponding set of valid segments, i.e., $s(u_i, u_j) = s(\{u_i, u_j\})$.

Figure 3.2: The VG-graph for Table 3.1

Note that, if $\{u_i, u_j\}$ is a valid 2-group, there is *only* one edge between the corresponding two vertices u_i and u_j . It is impossible to have multiple edges between the same two nodes. That is, the edge is associated with the set of valid segments of $\{u_i, u_j\}$, not a single valid segment.

Consider D in Table 3.1. Assume that $max_dis = 10$, $min_dur = 3$ and $min_wei = 50\%$. We derive $\mathbb{G}_2$ of D (see Table 5.1), using AGP algorithm (with some modifications to store valid segments as it computes $\mathbb{G}_2$). The constructed VG-graph is shown in Figure 3.2.

Note that the construction of VG-graph only requires one scan of the original movement database, whether or not the memory is large enough. All we have to do is to scan the

movement database once to select valid 2-groups ($\mathbb{G}_2$) with their valid segments. This can be done by applying AGP on the movement database for only one iteration to find valid 2-groups. Of course, we need to modify AGP a little to store the valid segments, since the original AGP only compute the weight count without storing the set of valid segments. Once the set of valid 2-groups and their valid segments are found, the corresponding VG-graph is already there.

Assuming an adjacency list representation, the space required for storing a VG-graph can be determined by:

$$|VG-graph| = \alpha|V| + \beta|E| + |s(\mathbb{G}_2)| \quad (\text{Eq. 3.3})$$

where α and β are the space required for storing a vertex and an edge respectively, and $|s(\mathbb{G}_2)|$ is the space required for storing the valid segment lists. Note that, $|V| \leq M$, $|E| = |\mathbb{G}_2|$ and $|s(\mathbb{G}_2)| \leq 2\delta|\mathbb{G}_2|\lfloor \frac{N}{\min_dur+1} \rfloor$, where δ is the space required for storing one time stamp, and the number of valid segments for a valid 2-group is at most $\lfloor \frac{N}{\min_dur+1} \rfloor$. In addition, we define the *compression ratio* of a VG-graph as:

$$\text{compression ratio of } VG\text{-graph} = \frac{|VG\text{-graph}|}{|D|} \quad (\text{Eq. 3.4})$$

where $|D|$ denotes the size of the original movement database.

Theorem 3.2 *Given a movement database D and thresholds $\max_dis$, $\min_dur$ and $\min_wei$, the corresponding VG-graph contains complete information of D relevant to valid group pattern mining.*

Proof. In the VG-graph construction process, all the valid 2-groups, associated with their valid segments, are stored in the VG-graph. From Theorem 3.1, we know that if a k -group ($k \geq 2$) pattern is valid, then all of its 2-subgroup patterns are valid as well. That is to say, each valid k -group can be generated from some valid 2-groups. Moreover, we can check the validity of a k -group by examining the *intersections* between the valid segments of all its 2-subgroups. Thus this Theorem holds. ■

VG-growth Algorithm

Based on the constructed VG-graph, we propose the VG-growth algorithm to efficiently mine all valid k -groups ($k > 2$). We first introduce the following definitions which are important to VG-growth algorithm.

Definition 3.7 *Given a vertex u_i in VG-graph (V, E, s) , the set of **suffix-neighbors** of u_i is defined as $\{u_j \mid u_j \in V, (u_i, u_j) \in E, j > i\}$.*

That is, the suffix-neighbors of a vertex u are the vertices which are neighbors of u and has larger user id than u .

Definition 3.8 *A **prefix-group**, denoted by H , is a list of valid users ordered by ascending user ids.*

In particular, a prefix-group can be empty, which is the initial case in VG-growth algorithm.

Definition 3.9 *The k -order **conditional VG-graph** of a prefix-group $H' = \{u_{a_1}, u_{a_2}, \dots, u_{a_k}\}$ ($a_1 < a_2 < \dots < a_k$) is denoted by $VG^k(H') = (V^k, E^k, s^k)$ and is constructed as follows.*

- (i) *When $k = 0$. $H' = \emptyset$ and $VG^0(\emptyset)$ (i.e., the 0-order conditional VG-graph) is the VG-graph constructed from $\mathbb{G}_2$.*
- (ii) *When $k \geq 1$. Let $VG^{k-1}(H) = (V^{k-1}, E^{k-1}, s^{k-1})$ denote the $(k-1)$ -order conditional VG-graph for $H = \{u_{a_1}, u_{a_2}, \dots, u_{a_{k-1}}\}$. Then, $VG^k(H')$ can be constructed from $VG^{k-1}(H)$ as follows:*

$$E^k = \{(u_i, u_j) \mid u_i, u_j \in V_{H'}, (u_i, u_j) \in E^{k-1}, |s^k(u_i, u_j)| \geq \text{min_wei} \cdot N\} \quad (\text{Eq. 3.5})$$

$$V^k = \{u_i \mid \exists u_j, s.t., (u_i, u_j) \in E^k\} \quad (\text{Eq. 3.6})$$

where

$$V_{H'} = \{u_i \mid (u_i, u_{a_k}) \in E^{k-1}, i > a_k\}; \quad (\text{Eq. 3.7})$$

$$s^k(u_i, u_j) = \{s \mid s \in s_{\cap i, j, a_k}, |s| \geq \text{min_dur}\}; \text{ and} \quad (\text{Eq. 3.8})$$

$$s_{\cap i, j, a_k} = s^{k-1}(u_i, u_j) \cap s^{k-1}(u_i, u_{a_k}) \cap s^{k-1}(u_j, u_{a_k}) \quad (\text{Eq. 3.9})$$

Algorithm 3.3 VG-growth Algorithm.**Input:** D , max_dis , min_dur , and min_wei **Output:** $\mathbb{G}$ **Steps:**

- 1: $\mathbb{G} = \emptyset$;
- 2: use AGP to find $\mathbb{G}_2$ and $s(\mathbb{G}_2)$;
- 3: construct VG-graph $VG^0(\emptyset) = (V^0, E^0, s^0)$ based on Definition 3.6;
- 4: **VG-growth**($VG^0(\emptyset)$, $\emptyset$, $\mathbb{G}$);
- 5: **return** $\mathbb{G}$;

Function **VG-growth**(VG-graph, H , $\mathbb{G}$) // $VG\text{-graph} = (V, E, s)$

- 1: **for** each $u_i \in V$ **do** // in user id increasing order
- 2: $H' = H \cup \{u_i\}$;
- 3: $V_{H'} = \{u_j \mid u_j \in V, (u_i, u_j) \in E, j > i\}$;
- 4: **if** $V_{H'} \neq \emptyset$ **then**
- 5: **for** each vertex $u_j \in V_{H'}$ **do**
- 6: obtain a valid group: $G = H' \cup \{u_j\}$;
- 7: $\mathbb{G} = \mathbb{G} \cup \{G\}$;
- 8: **end for**
- 9: $VG^{|H'|}(H') = \text{the conditional VG-graph of } H'$;
- 10: **VG-growth**($VG^{|H'|}(H')$, H' , $\mathbb{G}$);
- 11: **end if**
- 12: **end for**

The complete VG-growth algorithm is shown in Algorithm 3.3. Note that, VG-growth start from finding the set of valid 2-groups and the corresponding valid segments, i.e., $\mathbb{G}_2$ and $s(\mathbb{G}_2)$, by algorithm AGP⁴ (see line 02 in Algorithm 3.3). That is, both AGP and VG-growth share the same procedure of finding $\mathbb{G}_2$. VG-growth then involves a recursive call to a function, called VG-growth, using two variables H and $VG^{|H|}(H)$. H is a prefix-group with the initial status as $\emptyset$ and $VG^{|H|}(H)$ is the corresponding conditional VG-graph of H , with the initial status as the original VG-graph constructed from $\mathbb{G}_2$, i.e., $VG^0(\emptyset)$. Function VG-growth traverses each vertex in $VG^{|H|}(H)$ in user id increasing order and performs the following operations:

- (i) For each vertex u_i currently visited, extend H to H' by adding u_i , i.e., $H' = H \cup \{u_i\}$
- (ii) Select the set of suffix-neighbors of u_i in $VG^{|H|}(H)$, denoted by $V_{H'}$
- (iii) For each suffix-neighbor u_j in $V_{H'}$, output a valid group $G = H' \cup \{u_j\}$

⁴The set of valid segments can be found by AGP with minor modifications.

- (iv) Construct the conditional VG-graph of H' , i.e., $VG^{|H'|}(H')$
- (v) Recursively call function VG-growth using H' and $VG^{|H'|}(H')$

Suppose we apply VG-growth to the movement database shown in Table 3.1 with $max_dis = 10$, $min_dur = 3$, and $min_wei = 50\%$. As shown in Figure 3.3, VG-growth finds the same set of valid groups as AGP by performing a depth-first traversal on the search tree and generating a valid group for each node with depth ≥ 2 . Figure 3.4 presents the

Figure 3.3: Search Tree for VG-growth

details of the generation of the 28th to 33rd valid groups in Figure 3.3. Note that, in Figure 3.4, based on Equations Eq. 3.8 and Eq. 3.9, $s^1(\{u_4, u_5\}) = s^0(\{u_4, u_5\}) \cap s^0(\{u_3, u_4\}) \cap s^0(\{u_3, u_5\}) = [0, 9] \cap [0, 7] \cap [0, 7] = [0, 7]$.

Lemma 3.1 *Let $VG^k(H)$ be the conditional VG-graph of a prefix-group $H = \{u_{a_1}, \dots, u_{a_k}\}$ ($k \geq 0$, $a_1 < a_2 < \dots < a_k$), then every edge (u_i, u_j) in $VG^k(H)$ represents a valid $k + 2$ group $\{u_{a_1}, \dots, u_{a_k}, u_i, u_j\}$.*

Proof. We prove this lemma by induction.

[Base Case]. When $k = 0$, $VG^{(0)}(\emptyset)$ is the original VG-graph. It is clear that every edge (u_i, u_j) in the original VG-graph represents a valid 2-group $\{u_i, u_j\}$ and the base case holds.

Figure 3.4: An Illustrative Example of VG-growth

[*Inductive Hypothesis*]. Suppose the lemma holds for $k = n$. That is, every edge (u_i, u_j) in $VG^{(n)}(H) = (V^n, E^n, s^n)$ with $H = \{u_{a_1}, \dots, u_{a_n}\}$ represents a valid $n + 2$ group $\{u_{a_1}, \dots, u_{a_n}, u_i, u_j\}$.

[*Inductive Step*]. Consider $VG^{(n+1)}(\{u_{a_1}, \dots, u_{a_n}, u_{a_{n+1}}\}) = (V^{n+1}, E^{n+1}, s^{n+1})$. Based on the definition of conditional VG-graph, for each edge $(u_i, u_j) \in E^{n+1}$, there must exist (u_i, u_j) , $(u_i, u_{a_{n+1}})$, and $(u_j, u_{a_{n+1}})$ in E^n . Given the inductive hypothesis, these three edges represent three valid $n + 2$ groups: $\{u_{a_1}, \dots, u_{a_n}, u_i, u_j\}$, $\{u_{a_1}, \dots, u_{a_n}, u_{a_{n+1}}, u_i\}$, and $\{u_{a_1}, \dots, u_{a_n}, u_{a_{n+1}}, u_j\}$, denoted by G^I , G^{II} , and G^{III} respectively. In addition, the valid segments of G^I , G^{II} , and G^{III} are $s^n(u_i, u_j)$, $s^n(u_i, u_{a_{n+1}})$, and $s^n(u_j, u_{a_{n+1}})$ respectively.

Next, from Equations Eq. 3.8 and Eq. 3.9, we know that $s^{n+1}(u_i, u_j)$ satisfies *min_dur* and thus $s^{n+1}(u_i, u_j)$ is the valid segments of group $G^{IV} = G^I \cup G^{II} \cup G^{III}$. From Equation Eq. 3.5, $s^{n+1}(u_i, u_j)$ also satisfies *min_wel*, thus, G^{IV} is valid. That is, each edge (u_i, u_j) in $VG^{(n+1)}(H \cup \{u_{a_{n+1}}\})$ represents a valid $n + 3$ group $\{u_{a_1}, \dots, u_{a_n}, u_{a_{n+1}}, u_i, u_j\}$.

Thus, the lemma holds for $k = n + 1$. ■

Theorem 3.3 [*Correctness of VG-growth*]: *VG-growth only generates valid groups.*

Proof. Without loss of generality⁵, let $G = \{u_{a_1}, \dots, u_{a_k}\}$ be a valid group generated by VG-growth, where $a_1 < a_2 < \dots < a_k$. According to the mining process of VG-growth, G is generated when visiting vertex $u_{a_{k-1}}$ in $VG^{(k-2)}(H)$ where $H = \{u_{a_1}, \dots, u_{a_{k-2}}\}$, and there is an edge $(u_{a_{k-1}}, u_{a_k})$ in $VG^{(k-2)}(H)$. Based on Lemma 3.1, we know that this edge represents a valid k -group $\{u_{a_1}, \dots, u_{a_k}\}$. Thus, this property is proved. ■

Theorem 3.4 [*Completeness of VG-growth*]: *VG-growth generates all valid groups.*

Proof. Let $G = \{u_{a_1}, \dots, u_{a_k}\}$ be *any* valid group, where $a_1 < a_2 < \dots < a_k$ and $k \geq 2$. We need to prove that VG-growth will generate G as a valid group. We prove this in two cases: (1) $k = 2$; and (2) $k \geq 3$ as follows.

Case 1: ($k = 2$) G will be generated during mining the set of valid 2-groups using AGP (see Line 02 in Algorithm 3.3).

Case 2: ($k \geq 3$) Given that $G = \{u_{a_1}, \dots, u_{a_k}\}$ is a valid group, we know that $u_{a_1}, \dots, u_{a_k}$ are valid users in the original VG-graph. Based on the definition of conditional VG-graph, there must exist a complete subgraph formed by $u_{a_{i+1}}, \dots$, and u_{a_k} in $VG^{(i)}(H)$ with $H = \{u_{a_1}, \dots, u_{a_i}\}$, $\forall i, 2 \leq i < k$.

Considering the case when $i = k - 2$, there must be an edge $(u_{a_{k-1}}, u_{a_k})$ in $VG^{(k-2)}(\{u_{a_1}, \dots, u_{a_{k-2}}\})$. Therefore, when visiting vertex $u_{a_{k-1}}$, G will be generated as a valid group by VG-growth. Thus, this Theorem is proven. ■

3.3 Performance Evaluation

In this section, we evaluate and compare the performance of AGP and VG-growth.

3.3.1 Experimental Design and Data Generation

The experiments have been conducted using movement databases generated by IBM City Simulator [KMJ01] on a Pentium-IV machine with a CPU clock rate of 2.4 GHz and 1 GB of main memory. Note that both AGP and VG-growth were implemented assuming that

⁵Although there is no implicit ordering among the users of a group, we can always sort them by user ids.

the movement database resides in main memory. This reduces the required time for our experiments.

City Simulator can generate realistic three-dimensional user movement over city layout that includes streets and buildings. A dataset M1kN1k that contains 1000 users and 1000 time points was generated, covering a $1000m \times 1500m \times 100m$ area of 48 roads and 72 buildings with different heights (the highest is around 90 meters).

We recorded the total execution time (T), the time for mining valid 2-groups (T_2), and the time for mining all other valid groups (T_k) for different min_wei values ranging from 1% to 10%. $T = T_2 + T_k$ as both AGP and VG-growth find valid 2-groups first before the rest. The max_dis and min_dur thresholds were 30 and 4 respectively. In the experiments, the unit of measurement adopted for distance was meter, and the interval between every two consecutive time points represents 10 minutes. Thus, $N = 1000$ and $min_dur = 4$ represent about one week and 40 minutes respectively. Table 3.3 summarizes the parameters used in this set of experiments.

Table 3.3: Parameter Setting: AGP vs. VG-growth

Dataset	M	N	Dataset Size(MB)	Thresholds
M1kN1k	1000	1000	12	$max_dis = 30, min_dur = 4$ $min_wei = 1\%, 2\%, 4\%, 6\%, 10\%$

3.3.2 Performance Evaluation

As shown in Figure 3.5(a), VG-growth outperformed AGP in execution time T , especially when min_wei is small ($< 4\%$). In particular, when $min_wei = 1\%$, VG-growth runs 10 times faster than AGP. Figure 3.5(b) shows that the execution time differences come from T_k , since AGP and VG-growth share the same procedure of mining valid 2-groups. For small min_wei , VG-growth was more efficient than AGP due to larger number of valid groups with size > 2 that can be mined using VG-graph. In contrast, AGP suffered from large number of candidate groups, causing large overhead in database scans and validity checking of candidate groups.

As the min_wei increased, T_2 began to dominate T due to larger proportions of valid 2-groups, as shown in Figure 3.5(c). For example, when $min_wei = 10\%$, most valid groups

(a). T (M1kN1k).

(b). T₂ and T_k (M1kN1k). (c). |G| and |G₂| (M1kN1k).

(d). Number of Vertices in VG-graph.

(e). Size of VG-graph.

(f). Size of VG-graph v.s Original DB Size

Figure 3.5: Experiment Results: AGP vs. VG-growth

were of size 2 and both AGP and VG-growth spent almost all the time finding valid 2-groups. In other words, the VG-graph has little use to improve the execution time. This also motivates our proposed summarization approach to mine valid 2-groups which will be described in the next chapter.

In the experiment, we implemented the VG-graph using adjacency list structure. Each vertex in the VG-graph was stored with a list of its suffix-neighbors and the corresponding valid segment lists. Vertex ids and the time stamps were represented as 4 bytes integers and each list pointer required 4 bytes. The byte size of the VG-graph was obtained accordingly. As shown in Figure 3.5(d), the number of vertices in the VG-graph was the about same as the number of users when min_wei was less than 6%. It decreased with increasing min_wei . For example, when $min_wei = 10\%$, there were only 618 valid users (i.e., vertices) among the 1000 users in M1kN1k. Figure 3.5(e) shows the size of VG-graph in KB for different min_wei values. Our experiments have shown that the compression ratio of VG-graph for dataset M1kN1k, which occupies 12MB space on hard disk, was between $1\% \sim 6\%$, as shown in Figure 3.5(f). This indicates very good compression ratios achieved by VG-graph as it only contains the set of valid users and only stores the valid segments of each valid 2-group rather than the actual location records of each user.

The scalability results of VG-growth with respect to M and N are shown in Figures 3.6(a), 3.6(b) and 3.6(c). We measured both the execution time and the size of VG-graph for different numbers of users (M) and time points (N) as shown in 3.4.

Table 3.4: Parameter Setting: Scalability of VG-growth

M (thousands)	N (thousands)	Dataset Size(MB)	Thresholds
1	1/3/5/7/10	12 $\sim$ 120	$max_dis = 30, min_dur = 4$
1/3/5/7/10	1	12 $\sim$ 120	$min_wei = 1\%, 2\%, 4\%, 6\%, 10\%$

We only provide the curves for $min_wei = 4\%$, since the curves for other min_wei 's have the same trends. Figure 3.6(a) shows the scalability of VG-graph when M or N changes. The size of VG-graph increased almost quadratically with M but increased very little with N . This is because the increase of M leads to not only more vertices in the VG-graph but also more valid segments. On the other hand, the increase of N only causes more valid segments while the number of vertices increases only a little.

(a). Size of VG-graph: M/N Changes.

(b). Scale-up with N (VG-growth).

(c). Scale-up with M (VG-growth).

Figure 3.6: Experiment Results: Scalability of VG-growth

From Figures 3.6(b) and 3.6(c), we find that T increased almost linearly with N and almost quadratically with M . This is due to the dominating T_2 which has $O(N \cdot M^2)$ time complexity.

3.4 Summary

In this chapter, we formally define the group pattern mining problem and developed two algorithms, AGP and VG-growth, for mining the complete set of valid group patterns. The performance of the two algorithms has been evaluated using synthetically generated user movement databases. It has been shown that VG-growth is around an order of magnitude faster than AGP for small min_wei . Another observation is that the cost of mining group

patterns of size two is a dominant component of the total running time. Hence, the performance gain of VG-growth algorithm is most apparent when the *min_wei* is small, resulting in more valid groups of larger sizes. Moreover, considering that VG-graphs can be stored beforehand and its construction cost can be amortized over multiple runs of group pattern mining, the cost savings of VG-growth algorithm will be more significant.

Both VG-growth and AGP have identical steps for mining valid 2-groups, which is a costly operation. In order to reduce this overhead, we therefore propose, in the next chapter, a group pattern mining framework that can accommodate different location summarization methods to reduce both the number of time points to be scanned and the number of candidate valid 2-groups to be examined. Under this framework, several location summarization methods will be introduced.

Chapter 4

Efficiently Mining Valid 2-Groups by Location Summarization

As mentioned in the previous chapter, both AGP and VG-growth share the same step of mining valid 2-groups, i.e., scanning the database D to accumulate the weight count for each candidate use pair. Assume there are M distinct users and N time points in D . Therefore, there are $\binom{M}{2}$ candidate user pairs need to be examined. Hence, the time required to compute valid 2-groups consists of scanning $M \times N$ user location records, and determining the distances between $\binom{M}{2}$ pairs of users. In order to reduce such a overhead, we propose the following simple but effective data summarization method, which will find the same set of valid 2-groups as AGP or VG-growth does but with much less execution time.

4.1 Framework for Mining Valid 2-Groups Using Location Summarization

In this chapter, we propose to address the overhead of mining valid 2-groups. We first describe a common framework to incorporate different location summarization methods into valid 2-group mining. Subsequently, we present several location summarization methods that adopt different summarization models and assumptions on the summarization parameters.

The proposed framework consists of two steps: *preprocessing* and *mining*. In the preprocessing step, each user's movement data are first divided into time windows of the same size and locations within each time window are summarized using a *summarization model*. After that, the upper bounds of weight count and valid segment length for each user pair are computed based on the summarized location data.

In the mining step, we first generate a set of candidate 2-groups based on the upper bound information. This set of candidate 2-groups is expected to be smaller than all possible candidate 2-groups. Moreover, instead of scanning the large movement database, the much smaller summarized location database is scanned to check the validity of each candidate 2-group. Only when valid segments could not be determined based on the summarized data, the original database will then be accessed. We elaborate the details in the following subsections.

4.1.1 Preprocessing of User Movement Data

Figure 4.1: Example of Instances of Summarization Model

Let D'_i denote the summarized data of user u_i , in which the number of time points in the original movement data of u_i , D_i , is reduced to $N' = \lceil \frac{N}{w} \rceil$, where w is the time window size and N is the number of time points in D_i . For simplicity, we assume that $\frac{N}{w}$ is a whole number¹. Note that a time point t' in the summarized database D'_i corresponds to a time window $[t' \cdot w, (t' + 1) \cdot w)$ in D_i . We use $u_i[t'].P$ to denote $\{u_i[t].p \mid t' \cdot w \leq t < (t' + 1) \cdot w\}$. Based on a *summarization model* (or simply SM), which is some 3D geometry shape such as sphere, cube, etc., $u_i[t'].P$ is summarized to an *instance* of the corresponding SM, denoted by $sm(u_i, t')$. D'_i is therefore $\{sm(u_i, 0), sm(u_i, 1), \dots, sm(u_i, N' - 1)\}$. In addition, we

¹If not, the residual part can be simply treated as one time window. We can summarize the location values within it using the same method.

define the *summarization ratio* of a summarized database as $\frac{|D'|}{|D|}$, where $|D'|$ and $|D|$ are the sizes of D' and D respectively.

For example, Figure 4.1 illustrates the instances of summarization model, where N and w are 25 and 5 respectively. Note that, the location points within each time window are summarized into a sphere, i.e., an instance of the sphere summarization model, which will be described in detail later. The summarized database contains 5 spheres, each represented by a center and a radius.

In this research, we will examine four different SM's, namely:

- Sphere Location Summarization Method (SLS);
- Cuboid Location Summarization Method (CLS);
- Grid-Sphere Location Summarization Method (GSLs);
- Grid-Cuboid Location Summarization Method (GCLS);

These methods will be further described in Section 4.2. Extensions to GSLs and GCLS to consider maximum speed constraint on user movement are given in Appendix A. These extensions however yield performance results similar to GSLs and GCLS.

With D' , the number of time points to be scanned are reduced from N to $\frac{N}{w}$. However, this does not address the problem of scanning D' for large number of candidate 2-groups. Thus, in the preprocessing step, we pre-compute the *upper bounds* of weight count and valid segment length for each user pair based on D' . The pre-computation is carried out under the assumption that the upper bound of max_dis , denoted by $\overline{max_dis}$, is given.

Definition 4.1 Let t' be a time point in the summarized database D' . Let $\overline{max_dis}$ be the upper bound of max_dis (i.e., $\overline{max_dis} \geq max_dis$). Then $sm(u_i, t')$ and $sm(u_j, t')$ are said to be **possibly close**, if:

$$MinDistance(sm(u_i, t'), sm(u_j, t')) \leq \overline{max_dis} \quad (\text{Eq. 4.1})$$

where $MinDistance(sm(u_i, t'), sm(u_j, t'))$ is a function returning the minimum distance between $sm(u_i, t')$ and $sm(u_j, t')$.

Definition 4.2 Given a summarized database D' , a user pair $\{u_i, u_j\}$, and $\overline{max_dis}$, a set of consecutive time points $[t'_a, t'_b]$ is called a **possibly close segment (PCS)** of $\{u_i, u_j\}$, if:

- (i) $\forall t' \in [t'_a, t'_b]$, $sm(u_i, t')$ and $sm(u_j, t')$ are possibly close;
- (ii) If $t'_a > 0$, $sm(u_i, t'_a - 1)$ and $sm(u_j, t'_a - 1)$ are not possibly close;
- (iii) If $t'_b < N'$, $sm(u_i, t'_b + 1)$ and $sm(u_j, t'_b + 1)$ are not possibly close.

We use $S(\{u_i, u_j\})$ to denote the set of PCS's of $\{u_i, u_j\}$, i.e.,

$$S(\{u_i, u_j\}) = \{ [t'_a, t'_b] \mid [t'_a, t'_b] \subseteq [0, N'], [t'_a, t'_b] \text{ is a PCS of } \{u_i, u_j\} \} \quad (\text{Eq. 4.2})$$

Theorem 4.5 $\forall s \in s(\{u_i, u_j\}), \exists [t'_a, t'_b] \in S(\{u_i, u_j\})$ such that $s \subseteq [t'_a \cdot w, (t'_b + 1) \cdot w)$.

Proof. Recall that $s(\{u_i, u_j\})$ is the set of valid segments of $\{u_i, u_j\}$. Given any valid segment s of $\{u_i, u_j\}$, $s = [t_p, t_q]$ ($0 \leq t_p < t_q < N$), $[t_p, t_q]$ must lie *within* one time window of size w , or *across* more than one time windows, denoted by $[t'_m, t'_n]$. We have $t_p \geq (t'_m \cdot w)$ and $t_q < (t'_n + 1) \cdot w$. As u_i and u_j are not more than max_dis apart $\forall t \in [t_p, t_q]$, $sm(u_i, t'_k)$ and $sm(u_j, t'_k)$ should be possibly close at t'_k ($m \leq k \leq n$), since $\overline{max_dis} \geq max_dis$. Hence, we have (1) $[t'_m, t'_n]$ itself is a PCS of $\{u_i, u_j\}$, or (2) $[t'_m, t'_n]$ is covered by a PCS (say, $[t'_p, t'_q]$) of $\{u_i, u_j\}$. Let $[t'_a, t'_b]$ be $[t'_m, t'_n]$ (for case 1), or $[t'_p, t'_q]$ (for case 2). In both cases, we have $[t'_a, t'_b] \in S(\{u_i, u_j\})$, and $s = [t_p, t_q] \subseteq [t'_a \cdot w, (t'_b + 1) \cdot w)$. Therefore, this theorem holds. ■

The above theorem says $S(\{u_i, u_j\})$ consists of possibly close segments (in D') that cover all the valid segments of $\{u_i, u_j\}$ in D . This theorem provides the foundation of the correctness and completeness for the summarization based algorithms.

Definition 4.3 Given a user pair $\{u_i, u_j\}$, the **longest possibly close segment length** of $\{u_i, u_j\}$ is defined as:

$$Q(\{u_i, u_j\}) = w \cdot \underset{[t'_a, t'_b] \in S(\{u_i, u_j\})}{Max} (t'_b - t'_a + 1) \quad (\text{Eq. 4.3})$$

Theorem 4.6 $\forall s \in s(\{u_i, u_j\}), Q(\{u_i, u_j\}) \geq |s|$.

Proof. Let s_{max} be the longest valid segment of $\{u_i, u_j\}$. We want to show that $Q(\{u_i, u_j\}) \geq |s_{max}|$. Due to Theorem 4.5, there exists a PCS: $[t'_a, t'_b] \in S(\{u_i, u_j\})$ such that $s_{max} \subseteq [t'_a \cdot w, (t'_b + 1) \cdot w)$. Since $[t'_a, t'_b] \in S(\{u_i, u_j\})$, $Q(\{u_i, u_j\}) \geq w \cdot (t'_b - t'_a + 1) \geq |s_{max}|$.

Thus, the theorem is proven. ■

This theorem asserts that the longest possibly close segment length of a user pair is an upper bound of the valid segment length of this pair of users.

Definition 4.4 *The upper bound weight count of $\{u_i, u_j\}$ is defined as:*

$$R(\{u_i, u_j\}) = w \cdot \sum_{[t'_a, t'_b] \in S(\{u_i, u_j\})} (t'_b - t'_a + 1) \quad (\text{Eq. 4.4})$$

Theorem 4.7 $R(\{u_i, u_j\}) \geq \text{weight count}(\{u_i, u_j\})$.

Proof. Recall that $\text{weight count}(\{u_i, u_j\}) = \sum_{i=1}^n |s_i|$, where $s_i \in s(\{u_i, u_j\})$. Let $S(\{u_i, u_j\})$ be the set of PCS's of $\{u_i, u_j\}$. Note that, for any $\text{PCS} \in S(\{u_i, u_j\})$, there are two possible cases: (1) this PCS covers one or more valid segment(s); or (2) this PCS does not cover any valid segment, since $\overline{\text{max_dis}} \geq \text{max_dis}$. Let $S'(\{u_i, u_j\})$ denote the set of PCS's that covers one or more valid segment(s). Obviously, $S'(\{u_i, u_j\}) \subseteq S(\{u_i, u_j\})$.

Next, from Theorem 4.5, we know that, for each valid segment s_i , there exists a PCS covering s_i . Thus,

$$\sum_{i=1}^n |s_i| \leq w \cdot \sum_{\text{PCS} \in S'(\{u_i, u_j\})} |\text{PCS}|$$

From the definition of upper bound weight count, we know:

$$R(\{u_i, u_j\}) = w \cdot \sum_{\text{PCS} \in S(\{u_i, u_j\})} |\text{PCS}| \geq w \cdot \sum_{\text{PCS} \in S'(\{u_i, u_j\})} |\text{PCS}|$$

Therefore, $R(\{u_i, u_j\}) \geq \text{weight count}(\{u_i, u_j\})$. Thus, the theorem is proven. ■

This theorem asserts that the upper bound weight count of a user pair is indeed the upper bound on the weight count for this pair of users.

Let $\mathbb{P}$ denote the set of all user pairs together with their longest possibly close segment length and upper bound weight count, i.e.,

$$\mathbb{P} = \{ (\{u_i, u_j\}, Q(\{u_i, u_j\}), R(\{u_i, u_j\})) \mid 1 \leq i < j \leq M \} \quad (\text{Eq. 4.5})$$

where M is the number of distinct users.

$\mathbb{P}$ contains the pre-computed upper bounds information about the valid segment length and the weight count for each user pair. To efficiently find candidate 2-groups that satisfy the *min_dur* requirement, we sort $\mathbb{P}$ by Q value in *descending* order. We also use $(\mathbb{P}_k.c_2, \mathbb{P}_k.Q(c_2), \mathbb{P}_k.R(c_2))$ to denote the k th tuple in $\mathbb{P}$.

The detailed algorithm for the preprocessing step is shown in Algorithm 4.1.

Algorithm 4.1 Preprocessing Step of Framework

Input: D , w , and $\overline{max_dis}$
Output: D' and $\mathbb{P}$
Steps:

- 1: **for** each user pair $c_2 = \{u_i, u_j\}$ **do**
- 2: $Q(c_2) = R(c_2) = sum(c_2) = 0$;
- 3: **end for**
- 4: **for** ($t' = 0$; $t' < \frac{N}{w}$; $t'++$) **do**
- 5: **for** each user u_i **do**
- 6: $u_i[t'].P = \{u_i[t].p \mid t' \cdot w \leq t < (t' + 1) \cdot w\}$;
- 7: $sm(u_i, t') = \text{SummarizeLocation}(u_i[t'].P)$
- 8: add $sm(u_i, t')$ into D' ;
- 9: **end for**
- 10: **for** each user pair $c_2 = \{u_i, u_j\}$ **do**
- 11: **if** $\text{MinDistance}(sm(u_i, t'), sm(u_j, t')) \leq \overline{max_dis}$ **then**
- 12: $R(c_2) = R(c_2) + w$;
- 13: $sum(c_2) = sum(c_2) + w$;
- 14: **else**
- 15: **if** $sum(c_2) > Q(c_2)$ **then**
- 16: $Q(c_2) = sum(c_2)$;
- 17: **end if**
- 18: $sum(c_2) = 0$;
- 19: **end if**
- 20: **end for**
- 21: **end for**
- 22: sort $\mathbb{P}$ by $Q(c_2)$ in decreasing order;
- 23: **return** D' and $\mathbb{P}$;

4.1.2 Mining Valid 2-Groups

After the summarized database D' and precomputed upper bound information $\mathbb{P}$ are constructed, the mining step can be carried out to find the set of valid 2-groups, as shown in Algorithm 4.2. User specified *max_dis*, *min_dur* and *min_wei* are input to the mining step.

From $\mathbb{P}$, we first determine a set of candidate 2-groups, C_2 , such that for each $c_2 \in C_2$, $Q(c_2) \geq min_dur$ and $R(c_2) \geq min_wei \cdot N$ (see function *GetCandidate2Groups* in

Algorithm 4.2 Mining Step of Framework

Input: $D, D', max_dis, min_dur, min_wei, \mathbb{P}$, and w

Output: $\mathbb{G}_2$

Steps:

- 1: $\mathbb{G}_2 = \emptyset$;
- 2: $C_2 = \text{GetCandidate2Groups}(\mathbb{P}, min_dur, min_wei)$;
- 3: **for** ($t' = 0$; $t' < \frac{N}{w}$; $t'++$) **do**
- 4: **for** each candidate 2-group $c_2 \in C_2, c_2 = \{u_i, u_j\}$ **do**
- 5: $max_ij_dis = \text{MaxDistance}(sm(u_i, t'), sm(u_j, t'))$;
- 6: **if** $max_ij_dis \leq max_dis$ **then** $//u_i$ and u_j must be close
- 7: $c_2.cur_seg++ = w$;
- 8: **else**
- 9: $min_ij_dis = \text{MinDistance}(sm(u_i, t'), sm(u_j, t'))$;
- 10: **if** $min_ij_dis > max_dis$ **then** $//u_i$ and u_j must be far apart
- 11: **if** $c_2.cur_seg \geq min_dur$ **then**
- 12: $c_2.weight++ = c_2.cur_seg$;
- 13: **end if**
- 14: $c_2.cur_seg = 0$;
- 15: **else**
- 16: $\text{CheckOriginalDB}(D, t', w, c_2)$;
- 17: **end if**
- 18: **end if**
- 19: **end for**
- 20: **end for**
- 21: $\mathbb{G}_2 = \{c_2 \in C_2 \mid c_2.weight \geq min_wei \cdot N\}$;
- 22: **return** $\mathbb{G}_2$;

Function $\text{GetCandidate2Groups}(\mathbb{P}, min_dur, min_wei)$

- 1: **for** ($i = 0$; $i < |\mathbb{P}|$; $i++$) **do**
- 2: **if** $\mathbb{P}_i.Q(c_2) \geq min_dur$ **then**
- 3: **if** $\mathbb{P}_i.R(c_2) \geq min_wei \cdot N$ **then**
- 4: add $\mathbb{P}_i.c_2$ into C_2 ;
- 5: **end if**
- 6: **else**
- 7: **break**;
- 8: **end if**
- 9: **end for**
- 10: **return** C_2 ;

Function $\text{CheckOriginalDB}(D, t', w, c_2)$

- 1: **for** ($t = t' \cdot w$; $t < (t' + 1) \cdot w$; $t++$) **do**
- 2: **if** $d(u_i[t].p, u_j[t].p) \leq max_dis$ **then**
- 3: $c_2.cur_seg++$;
- 4: **else**
- 5: **if** $c_2.cur_seg \geq min_dur$ **then**
- 6: $c_2.weight = c_2.weight + c_2.cur_seg$;
- 7: **end if**
- 8: $c_2.cur_seg = 0$;
- 9: **end if**
- 10: **end for**

Algorithm 4.2). Note that, we will not omit any candidate 2-groups which may eventually become valid 2-groups. We only select those candidates whose Q values are no less than min_dur and R values are no less than $min_wei \cdot N$. Considering Q value is the longest possibly close segment length and R value is the upper bound weight count, we can make sure that only those candidate 2-groups that are unpromising to become valid are pruned.

Next, we compute the weight count of each $c_2 \in C_2$ by scanning the summarized database D' . We classify the closeness of two instances of SM at a summarized time point into three cases:

- Case 1: all location points within the two instances of SM are no more than max_dis apart, which indicates that the corresponding two users must be “close” ($\leq max_dis$) at each time point within this time window. Thus, we can safely increment their current segment length by w (see lines 6-7 in Algorithm 4.2).
- Case 2: all location points within the two instances of SM are more than max_dis apart, which indicates that the corresponding two users cannot be “close” ($\leq max_dis$) at any time point within this time window. Thus, we can safely move to the next time window, i.e., next t' . (see lines 10-14 in Algorithm 4.2).
- Case 3: otherwise, i.e., only some location points inside the two instances of SM are less than max_dis apart, which indicates that the corresponding two users are “close” ($\leq max_dis$) at *some* time points within this time window and are not close ($> max_dis$) at the other time points (see line 16 in Algorithm 4.2).

Should case 3 arises, the corresponding time window in the *original* movement database D will be examined to check the closeness for each single time point within this time window in order to correctly compute the weight count.

Time complexity analysis. In Algorithm 4.2, Line 2 generates the set of candidate 2-groups based on min_dur and min_wei . The time complexity of function *GetCandidate2Groups* is $O(k)$, where k is the number of pairs with $Q(c_2) \geq min_dur$. Note that, $|C_2| = k'$ ($k' \leq k$), where k' is the number of pairs that satisfies both $Q(c_2) \geq min_dur$ and $R(c_2) \geq min_wei \cdot N$.

Lines 3-20 compute the weight count for each candidate 2-group. The time cost of lines 3-20 is: $n_1 \cdot T_{Max} + n_2 \cdot (T_{Max} + T_{Min}) + n_3 \cdot (T_{Max} + T_{Min} + T_{COD})$, where n_1, n_2 and n_3 are

the number of times when the above three cases are encountered respectively. T_{Max} , T_{Min} , and T_{COD} are the time costs of functions *MaxDistance*, *MinDistance* and *CheckOriginalDB* respectively. Note that, $n_1 + n_2 + n_3 = N' \cdot |C_2|$ as there are altogether $N' \cdot |C_2|$ iterations and $T_{COD} = O(w)$. Thus, the time complexity of lines 3-20 is $O(n_1 + n_2 + w \cdot n_3)$.

Lines 21-22 select and output the set of valid 2-groups, which costs $O(|C_2|) = O(k')$.

Thus, the total time cost is:

$$O(k + n_1 + n_2 + w \cdot n_3 + k').$$

In the best case, $n_3 = 0$, the total time cost becomes:

$$O(k + N' \cdot k' + k') = O\left(\frac{N}{w} \cdot k'\right).$$

In the worst case, $n_3 = N' \cdot |C_2| = N' \cdot k'$, the total time cost becomes:

$$O(k + N' \cdot k' \cdot w + k') = O(N \cdot k').$$

As mentioned before, without the location summarization, the time cost for finding all valid 2-groups is $O(N \cdot \binom{M}{2})$. Note that, here $k' \leq k \leq \binom{M}{2}$, which indicates the location summarization based algorithms always outperform AGP and VG-growth for mining valid 2-groups.

4.2 Location Summarization Methods

In the following, we will introduce four location summarization methods. Each method has its own SM and the corresponding *SummarizeLocation*, *MinDistance*, and *MaxDistance* functions.

4.2.1 Sphere Location Summarization (SLS) Method

The sphere location summarization method adopts a *sphere* as the SM. Each sphere is represented by (p_c, r) , where p_c is the center and r is the radius. Given w location values from a time window, $u_i[t'].P$, we compute the *minimal* and *maximal* x -, y -, z - values, denoted by $u[t'].x_{min}$, $u[t'].x_{max}$, $u[t'].y_{min}$, $u[t'].y_{max}$, $u[t'].z_{min}$, and $u[t'].z_{max}$. The center and radius of the sphere at time t' are determined respectively by:

$$p_c = \left(\frac{u[t'].x_{min} + u[t'].x_{max}}{2}, \frac{u[t'].y_{min} + u[t'].y_{max}}{2}, \frac{u[t'].z_{min} + u[t'].z_{max}}{2} \right) \quad (\text{Eq. 4.6})$$

$$r = \max_{p \in u_i[t'].P} d(p, p_c) \quad (\text{Eq. 4.7})$$

Given two spheres (p_{c_i}, r_i) and (p_{c_j}, r_j) , the minimum and maximum distances between them can be easily computed as follows.

$$Mindistance = d(p_{c_i}, p_{c_j}) - (r_i + r_j) \quad (\text{Eq. 4.8})$$

$$Maxdistance = d(p_{c_i}, p_{c_j}) + (r_i + r_j) \quad (\text{Eq. 4.9})$$

4.2.2 Cuboid Location Summarization (CLS) Method

Figure 4.2: SLS vs. CLS

As shown in Figure 4.2, instead of using sphere, the cuboid location summarization (CLS) method uses a *cuboid* to represent locations within a time window. Considering the fact that most users travel larger distance in x-y plane than in the z- dimension, CLS may be more compact than SLS². A more precise summarization will result in: (1) fewer calls to function *CheckOriginalDB*; (2) fewer candidate 2-groups to be generated.

The cuboid is defined such that its edges are parallel to the x-/y-/z- axes and is represented by (v_{min}, v_{max}) , where v_{min} is the corner with minimum x-/y-/z- values, and v_{max} is that with maximum x-/y-/z- values.

The minimum and maximum distances between two cuboids (v_{min_i}, v_{max_i}) and (v_{min_j}, v_{max_j}) are determined as follows:

$$Mindistance = \sqrt{(d_x^{min})^2 + (d_y^{min})^2 + (d_z^{min})^2} \quad (\text{Eq. 4.10})$$

where

$$d_x^{min} = \begin{cases} 0 & \text{if } [v_{min_i}.x, v_{max_i}.x] \cap [v_{min_j}.x, v_{max_j}.x] \neq \emptyset \\ \max(v_{min_i}.x, v_{min_j}.x) - \min(v_{max_i}.x, v_{max_j}.x) & \text{otherwise} \end{cases} \quad (\text{Eq. 4.11})$$

²In cuboid location summarization, the edges of a cuboid are parallel to the axes.

The d_y^{min} and d_z^{min} values are defined similarly.

$$Maxdistance = \sqrt{(d_x^{max})^2 + (d_y^{max})^2 + (d_z^{max})^2} \quad (\text{Eq. 4.12})$$

where

$$\begin{aligned} d_x^{max} &= \max((v_{max_i}.x - v_{min_j}.x), (v_{max_j}.x - v_{min_i}.x)) \\ d_y^{max} &= \max((v_{max_i}.y - v_{min_j}.y), (v_{max_j}.y - v_{min_i}.y)) \\ d_z^{max} &= \max((v_{max_i}.z - v_{min_j}.z), (v_{max_j}.z - v_{min_i}.z)) \end{aligned} \quad (\text{Eq. 4.13})$$

4.2.3 Grid-Sphere Location Summarization (GSLs) Method

In both SLS and CLS, the actual coordinates and radius are used in location summarization. To further reduce the size of SM, grid based location summarization methods are introduced. One of the grid based methods to be introduced is Grid-Sphere Location Summarization (GSLs), which is a grid extension to SLS.

Grid based method partitions the space into a set of **cells**, or cubes, of length l , as shown in Figure 4.3. Each cell in the grid is given a unique id. Given a location (x, y, z) , the id of the corresponding cell c can be determined by:

$$c.id = c.x' + N_x \cdot c.y' + (N_x \cdot N_y) \cdot c.z' \quad (\text{Eq. 4.14})$$

where $c.x' = \lfloor \frac{x}{l} \rfloor$, $c.y' = \lfloor \frac{y}{l} \rfloor$, $c.z' = \lfloor \frac{z}{l} \rfloor$, $N_x = \lceil \frac{XMAX}{l} \rceil$, $N_y = \lceil \frac{YMAX}{l} \rceil$ and $XMAX/YMAX$ are the maximum values in x-/y- dimensions respectively.

Conversely, given a cell index $c.id$, the corresponding x' , y' , z' index values can be obtained by a **Reverse** function, i.e., $(c.x', c.y', c.z') = \text{Reverse}(c.id, N_x, N_y)$.

$$\begin{aligned} c.z' &= \frac{c.id}{N_x \cdot N_y} \\ c.y' &= \frac{c.id - N_x \cdot N_y \cdot c.z'}{N_x} \\ c.x' &= c.id - N_x \cdot N_y \cdot c.z' - N_x \cdot c.y' \end{aligned} \quad (\text{Eq. 4.15})$$

The main idea of grid based method is to use a single $c.id$ instead of the actual coordinates to represent a location point. This results in large storage saving for D' . Furthermore, the tradeoff between size and accuracy of summarization can be easily tuned by cell length.

Figure 4.3: Partition 3D Space into Cells

In GSLS, the locations within a time window are summarized into a sphere represented by $(p_{c.id}, \bar{r})$, where $p_{c.id}$ is the *center cell id* and $\bar{r}$ is the *discretized radius*. The center cell id refers to the cell in which the actual center of the sphere is found. The discretized radius $\bar{r}$ is defined by $\lceil \frac{r}{\gamma} \rceil$, where r is the original radius and γ is the *radius scale unit* for discretizing radius. Such a sphere is also known as the *grid-sphere*.

Given two grid-spheres $(p_{c_i.id}, \bar{r}_i)$ and $(p_{c_j.id}, \bar{r}_j)$, the minimum and maximum distances between them can be determined by first deriving two larger non-grid spheres (p'_{c_i}, r'_i) and (p'_{c_j}, r'_j) that contain the two original grid-spheres, i.e.,

$$\begin{aligned}
 p'_{c_i} &= l \cdot (p_{c_i}.x' + \frac{1}{2}, p_{c_i}.y' + \frac{1}{2}, p_{c_i}.z' + \frac{1}{2}) \\
 p'_{c_j} &= l \cdot (p_{c_j}.x' + \frac{1}{2}, p_{c_j}.y' + \frac{1}{2}, p_{c_j}.z' + \frac{1}{2}) \\
 r'_i &= \bar{r}_i \cdot \gamma + \frac{\sqrt{3}}{2}l \\
 r'_j &= \bar{r}_j \cdot \gamma + \frac{\sqrt{3}}{2}l
 \end{aligned} \tag{Eq. 4.16}$$

That is, the centers of the two larger non-grid spheres, i.e., p'_{c_i} and p'_{c_j} , are the geometrical centers of the two cells $p_{c_i}.id$ and $p_{c_j}.id$ respectively. The radii r'_i and r'_j are derived from the discretized radii and the radius scale unit. Note that, the derived radii are augmented by $\frac{\sqrt{3}}{2}l$ in order to ensure that the non-grid spheres cover the original grid-spheres, since the maximal offset between the old and new centers is the distance from the new center

to the corner of the cell, which is just $\frac{\sqrt{3}}{2}l$. In addition, the x' , y' , z' index values can be obtained by Reverse function.

Then, the minimum and maximum distances are computed based on (p'_{c_i}, r'_i) and (p'_{c_j}, r'_j) , i.e.,

$$\begin{aligned} Mindistance &= d(p'_{c_i}, p'_{c_j}) - (r'_i + r'_j) \\ Maxdistance &= d(p'_{c_i}, p'_{c_j}) + (r'_i + r'_j) \end{aligned} \quad (\text{Eq. 4.17})$$

4.2.4 Grid-Cuboid Location Summarization (GCLS) Method

Similar to GSLS, GCLS summarizes the locations within each time window into a cuboid such that the cuboid consists of grid cells and is represented by the ids of the two cells that contains the locations with the minimum x-/y-/z- values and maximum x-/y-/z- values. This cuboid is also known as the *grid-cuboid*. Each grid-cuboid is denoted by $\langle id^{min}, id^{max} \rangle$. For user u_i at t' , the corresponding id^{min} and id^{max} are determined as:

$$id^{min} = \lfloor \frac{u_i[t'].x_{min}}{l} \rfloor + N_x \cdot \lfloor \frac{u_i[t'].y_{min}}{l} \rfloor + (N_x \cdot N_y) \cdot \lfloor \frac{u_i[t'].z_{min}}{l} \rfloor \quad (\text{Eq. 4.18})$$

$$id^{max} = \lfloor \frac{u_i[t'].x_{max}}{l} \rfloor + N_x \cdot \lfloor \frac{u_i[t'].y_{max}}{l} \rfloor + (N_x \cdot N_y) \cdot \lfloor \frac{u_i[t'].z_{max}}{l} \rfloor \quad (\text{Eq. 4.19})$$

Similar to GSLS, given two grid-cuboids $\langle id_i^{min}, id_i^{max} \rangle$ and $\langle id_j^{min}, id_j^{max} \rangle$, the minimum and maximum distances between them can be determined by first deriving two larger non-grid cuboids, $(v'_{i_{min}}, v'_{i_{max}})$ and $(v'_{j_{min}}, v'_{j_{max}})$, containing the two original grid-cuboids, i.e.,

$$\begin{aligned} v'_{i_{min}} &= l \cdot (id_i^{min}.x', id_i^{min}.y', id_i^{min}.z') \\ v'_{i_{max}} &= l \cdot (id_i^{max}.x' + 1, id_i^{max}.y' + 1, id_i^{max}.z' + 1) \\ v'_{j_{min}} &= l \cdot (id_j^{min}.x', id_j^{min}.y', id_j^{min}.z') \\ v'_{j_{max}} &= l \cdot (id_j^{max}.x' + 1, id_j^{max}.y' + 1, id_j^{max}.z' + 1) \end{aligned} \quad (\text{Eq. 4.20})$$

The x' , y' , z' index values can be obtained by Reverse function. Then, the minimum and maximum distances are computed based on $(v'_{i_{min}}, v'_{i_{max}})$ and $(v'_{j_{min}}, v'_{j_{max}})$ by using Equations Eq. 4.10-Eq. 4.13.

4.2.5 Summary of Location Summarization Methods

We summarize the location summarization methods in Table 4.1, in which the columns from left to right in turn are the location summarization method, the number of users in the summarized database, the number of time points in the summarized database, the summarization model, and the size of the summarized database (in terms of bits). Note that, the last row in Table 4.1 represents the movement database used by AGP or VG-growth without location summarization. In Table 4.1, η , $|r|$, $|\bar{r}|$, and $|id|$ denote the number of bits required to represent a coordinate value, a radius, a discretized radius, and a cell id respectively.

Table 4.1: Summary of Location Summarization Methods

Method	# of Users	# of Time Points	SM	Summarized Database Size (bits)
SLS	M	$\frac{N}{w}$	(p_c, r)	$(3\eta + r)M\frac{N}{w}$
CLS			(v_{min}, v_{max})	$6\eta M\frac{N}{w}$
GSLs			$(p_c, id, \bar{r})$	$(id + \bar{r})M\frac{N}{w}$
GCLS			(id^{min}, id^{max})	$2 id M\frac{N}{w}$
AGP&VG-growth	M	N	$\langle x, y, z \rangle$	$3\eta MN$

4.3 Performance Evaluation of Location Summarization Based Algorithms

In this set of experiments, we evaluate the performance of the location summarization based algorithms for mining valid 2-groups using different summarization methods.

4.3.1 Experimental Design and Data Generation

The experiments have been conducted using movement databases generated by IBM City Simulator on a Pentium-IV machine with a CPU clock rate of 2.4 GHz and 1 GB of main memory. Throughout this set of experiments, the dataset used is M1kN10k, which contains 1000 users and 10,000 time points and we fixed $max_dis = 30$, $min_dur = 4$, and $min_wei = 1\%$.

We first evaluate the impact of summarization parameters on the performance. We also compare the performance of the different algorithms when the summarized database size is

fixed. Finally, we compare the performance between AGP and the location summarization based algorithms.

Recall that, in the experiments described in Section 3.3, the movement database was loaded into main memory and mining did not involve hard disk access. In this set of experiments, however, only the summarized database was loaded into main memory, while the original movement database resided on the hard disk. The intention is to simulate the situation where the movement database is too large to be loaded into memory and to investigate the performance of the location summarization methods under such a disadvantaged condition. It is obvious that if the movement database is also loaded into memory, the performance of the location summarization based algorithms will be even better than that presented here.

4.3.2 Performance Evaluation

Impact of time window size

In this experiment, we studied the influence of the time window size w on the execution time of the group pattern mining algorithm using different location summarization methods.

Different w valued were chosen: 4, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 100, 200, and 400. The upper bound of max_dis threshold (i.e., $\overline{max_dis}$) was chosen as 40. For SLS and CLS, w is the only summarization parameter. On the other hand, GSLS and GCLS may have multiple combinations of $|id|$ and $|\overline{r}|$ values for a single w value. We therefore ran GSLS and GCLS with different combinations of parameters for each w and reported the ones that gave the smallest T_2 . The results are shown in Figure 4.4(a), in which we only plot the T_2 curves of SLS and CLS as the curve of GSLS is almost identical to that of SLS. The same applies to GCLS and CLS.

Intuitively, if w is very small, each instance of SM becomes more precise because of fewer location points within a time window. However, a large $N' = \frac{N}{w}$ results in more overhead to scan the summarized database. On the other hand, when w is very large, $N' = \frac{N}{w}$ becomes smaller. But the summarization is relatively coarse and the minimum distance between two instances of SM at any summarized time point is likely to be less than $\overline{max_dis}$. This causes large number of candidate 2-groups to be generated and more mining overhead. Therefore, there exist some optimal time window size between the two extremes.

As expected, we can see from Figure 4.4(a), the performance of SLS and CLS does not scale up linearly with w . In fact, T_2 decreases first when w increases from 4 to around 30 for CLS and 20 for SLS. After that, T_2 increases when w increases further. Each summarization method has an optimal w value. For SLS and GSLS, such an optimal w is around 24. For CLS and GCLS, the optimal w is around 32.

In order to understand the reason behind this observation, we decompose T_2 into $T_{case1,2}$ and T_{case3} as follows: $T_2 = T_{case1,2} + T_{case3}$, where $T_{case1,2}$ is the time spent on Case 1 and Case 2 (see Page 57) and T_{case3} is the time spent on Case 3. We have:

$$T_{case1,2} = n_1 \cdot T_{Max} + n_2 \cdot (T_{Max} + T_{Min}) = O(n_1 + n_2)$$

$$T_{case3} = n_3 \cdot (T_{Max} + T_{Min} + T_{COD}) = O(n_3 + w \cdot n_3).$$

Note that, $T_{case1,2}$ is solely memory based while T_{case3} consists of two parts. The first part “ n_3 ” represents the memory based cost and the second part “ $w \cdot n_3$ ” represents the hard disk based cost, i.e., the time used for checking the original database which resides on hard disk. Intuitively, the hard disk accessing cost would dominate T_2 .

Figure 4.4(b) shows $T_{case1,2}$ and T_{case3} for SLS. The other summarization methods produce similar curves and we therefore chose not to show them. As w increases, $T_{case1,2}$ becomes negligible, while T_{case3} becomes almost identical to T_2 , since T_{case3} dominates T_2 . Furthermore, from Figure 4.4(c), we can see those components affecting $T_{case1,2}$ and T_{case3} . Note that, both the x - and y - axes have logarithmic scale. As w increases, the total number of iterations for mining valid 2-groups, $N' \cdot |C_2|$, decreases. Both $n_1 + n_2$ and n_3 also decrease with increasing w . The former results in smaller $T_{case1,2}$, since $T_{case1,2} = O(n_1 + n_2)$. However, although n_3 decreases, $w \cdot n_3$ increases with the increasing of w . Recall that they represent memory based cost and hard disk based cost respectively. Hence, it is clear that the memory based cost overrides the hard disk based cost before reaching the optimal w , and reversely beyond the optimal w .

From this experiment, we also found that, with the same w , the two cuboid based methods, i.e., CLS and GCLS, always outperform the two sphere based methods, i.e., SLS and GSLS. This is because, cuboid based methods provide more precise summarizations than sphere based methods. This leads to smaller number of candidate 2-groups, $|C_2|$, and fewer accesses to the original database. We define the **summarization precision**, denoted by S_p , as:

$$S_p = \frac{1}{w \cdot \sum_{i=1}^M \sum_{t'=1}^{N'} V_{sm}(u_i, t')} \quad (\text{Eq. 4.21})$$

Figure 4.4: Impact of time window size

where $V_{sm(u_i, t')}$ is the *volume* of the geometrical shape formed by $sm(u_i, t')$.

We compared the summarization precision of different location summarization methods. We found that the summarization precision of SLS is slightly higher than that of GSLS. The same observation also applies to CLS and GCLS. Therefore, we only present the results of SLS and CLS here, as shown in Figure 4.4(d). Note that, instead of giving the absolute value, we provide the relative ratio, i.e., $\frac{S_p(CLS)}{S_p(SLS)}$. We can see that the summarization precision of CLS is much higher than that of SLS for any w value. In fact, the S_p of CLS is around 10-60 times of that of SLS. The difference becomes smaller when w increases because the compactness of CLS suffers when w becomes very large.

Impact of $\overline{max_dis}$

In this experiment, we studied the effect of $\overline{max_dis}$ on the mining overhead. For each time window size in the above experiments, $\overline{max_dis}$ was chosen as 30, 40, 60, 100 and 500. Figure 4.5(a) gives the curves for SLS and CLS for different $\overline{max_dis}$ values, in which we only show the curves for $w = 32$ as the curves for other w values have the same trend. Note that the curve of GSLS is almost identical to that of SLS. The same also applies to GCLS and CLS.

(a). T_2 . (b). $|C_2|$. (c). n_2 and n_3 .Figure 4.5: Impact of $\overline{max_dis}$

We can see that T_2 increases very little when $\overline{max_dis}$ increases. This may sounds

counterintuitive because we expect a larger $\overline{max_dis}$ to generate more candidate 2-groups that may significantly increase the mining overhead. Indeed, $|C_2|$ increases dramatically with $\overline{max_dis}$ increasing, as shown in Figure 4.5(b). However, a large $|C_2|$ alone is not sufficient to increase T_2 much, as T_2 is dominated by T_{case3} . Recall that $T_{case3} = O(n_3 + w \cdot n_3)$. With a fixed w , n_3 is a dominant factor. As shown in Figure 4.5(c), n_3 is almost constant when $\overline{max_dis}$ increases. The underlying reason is, although $|C_2|$ increases quickly causing more checking iterations during mining, most of them fall into Case 2 (see Page 57), with only very few iterations falling into Case 3.

In conclusion, within a certain range, e.g., less than 60 in our experiments, $\overline{max_dis}$ does not affect the execution time much for all the methods. Therefore, we simply fixed $\overline{max_dis}$ as 40 in the following experiments.

Performance Comparison with Fixed Summarization Database Size

Cuboid based algorithms outperforms sphere based algorithms with the same w . But, they generate summarized databases of different sizes. Thus, it is not a fair comparison. We therefore need to compare them based on the same summarized database size.

In this experiment, we chose seven different $\frac{|D'|}{|D|}$ ratios, ranging from 0.08% to 8.33%. In order to make w a whole number, we chose $\frac{|D'|}{|D|}$ as: $\frac{1}{1200}$, $\frac{1}{120}$, $\frac{1}{60}$, $\frac{1}{48}$, $\frac{1}{36}$, $\frac{1}{24}$, and $\frac{1}{12}$. For each $\frac{|D'|}{|D|}$ ratio, we run each algorithm for several parameter combinations. In particular, for SLS and CLS, $\frac{|D'|}{|D|}$ ratio can be used to determine w . On the other hand, GSLS and GCLS may adopt different combinations of w , $|id|$, and $|\overline{\tau}|$ values for the same $\frac{|D'|}{|D|}$ ratio. Therefore, for each $\frac{|D'|}{|D|}$ ratio, we run GSLS and GCLS with different combinations of parameters, and reported the ones that give the smallest T_2 .

As shown in Figure 4.6, the cuboid based methods still enjoy smaller T_2 than the sphere based methods. When the $|D'|$ is relatively large (e.g., $\frac{|D'|}{|D|} > 6\%$), the non-grid based methods perform slightly better than the grid based methods. The reason is, when $\frac{|D'|}{|D|}$ is large, SLS and CLS have more optimal w than GSLS and GCLS. For example, when $\frac{|D'|}{|D|} = \frac{1}{12}$ (8.33%), SLS and CLS have w values as 16 and 24 respectively. On the other hand, GSLS and GCLS share a common w , i.e., 8 (see Table B.2 in Appendix B). Note that the optimal w values for sphere and cuboid based methods are around 24 and 32 respectively. For the same reason, when the summarized database is small (e.g., $\frac{|D'|}{|D|} < 4\%$), the grid based algorithms have w values closer to optimal values than the non-grid based algorithms, e.g., when $\frac{|D'|}{|D|} = \frac{1}{120}$ (0.83%) (see Table B.1 in Appendix B), leading to smaller T_2 .

Figure 4.6: Performance Comparison with Fixed $|D'|$

Performance Comparison between AGP and Summarization based Mining Algorithms

In this part, we compare the performance of the algorithms for mining valid 2-groups using location summarization methods with AGP³. In particular, we combine the results from Sections 3.3 and 4.3.2 with the same parameter setting, i.e., dataset M1kN10k, $max_dis = 30$, $min_dur = 4$, and $min_wei = 1\%$.

Figure 4.7(a) gives the T_2 curves of the different algorithms. We can see that T_2 values of the location summarization based algorithms are around 10% – 20% of that of AGP. This illustrates the advantage by using location summarization based algorithms. Recall that AGP runs in main memory, while the location summarization based algorithms need to access the disk-resident original database D . Even so, the latter could still outperform AGP significantly. If the original database was also loaded into main memory, the T_2 of the location summarization based algorithms would be expected to be much smaller.

Figure 4.7(b) shows the size of candidate 2-groups generated by AGP and the location summarization based algorithms. Here, we only drew the curves for SLS and CLS, since GSLS and GCLS were very close (slightly below) to those of SLS and CLS respectively. Note that, AGP always generates a constant number of candidate 2-groups, i.e., $\binom{1000}{2} = 499500$. On the other hand, the location summarization based algorithms generate different sets

³Recall that AGP and VG-growth share the same procedure of finding the set of valid 2-groups.

Figure 4.7: Performance Comparison with AGP

of candidate 2-groups, which are much smaller than those generated by AGP. In fact, the ratio of $|C_2| / \binom{M}{2}$ for the location summarization based algorithms was around 13%. This significantly reduced the overhead for mining valid 2-groups.

4.4 Summary

In this chapter, we addressed the high overhead problem of mining valid 2-groups by AGP or VG-growth. We have introduced a group pattern mining framework that can accommodate different location summarization methods. Under this framework, we proposed several location summarization methods, including SLS, CLS, GSLS, and GCLS. These methods allow valid 2-groups to be mined using smaller number of summarized records and by examining smaller number of candidate 2-groups. We also conducted experiments to evaluate the performance of the proposed location summarization based methods. The experiment results have shown that our proposed location summarization based methods are much more efficient than AGP or VG-growth for mining valid 2-groups. The cuboid based methods also perform better than the sphere based methods.

Until now, we have addressed the problem of mining the exact set of valid groups. The solution is: first using location summarization based methods to facilitate the mining process for valid 2-groups, then constructing the VG-graph based on the mined valid 2-groups, finally applying VG-growth on the constructed VG-graph to mine all valid k -groups ($k > 2$).

However, we realized that the complete set of valid groups contains large redundant portion, which not only causes significant valid group mining overhead, but also compromises the usefulness of the mining results. Therefore, we study further to address this problem in the next chapter.

Chapter 5

Maximal Valid Group Mining

5.1 Problem Definition

Using VG-growth and the location summarization based algorithms, one can efficiently find the complete set of valid groups from a large movement database. However, the complete set of valid groups can be very large due to the exponential enumeration nature of all possible valid groups. In general, a group G of size k has $\binom{k}{2}$ subgroups¹ of size 2, $\binom{k}{3}$ subgroups of size 3, ..., $\binom{k}{k-1}$ subgroups of size $k - 1$. If G is a valid group, all of the $2^k - k - 2$ subgroups will also be valid. The mining of all valid groups not only causes significant valid group mining overhead, but also compromises the usefulness of mining results due to the large proportion of redundant valid groups.

In this chapter, we address the above redundancy problem by mining only the set of *maximal valid groups* defined below.

Definition 5.1 *Given a set of valid groups $\mathbb{G}$ and a valid group $X \in \mathbb{G}$, X is called a **maximal valid group**, if $\nexists Y \in \mathbb{G}$, s.t., $X \subset Y$.*

For example, given that $max_dis = 10$, $min_dur = 3$ and $min_wei = 50\%$, we can find 38 valid groups from the movement database shown in Table 3.1, as shown in Table 5.1. Among them, $\{u_2, u_3, u_4, u_5, u_6\}$ is a maximal valid group since it is not a subgroup of any other valid groups.

Definition 5.2 *Given the thresholds max_dis , min_dur , and min_wei , the problem of finding the exact set of maximal valid groups is known as **maximal valid group mining**.*

¹Please note that a single user is not considered to be a 1-group since a proper group must have at least two members.

Table 5.1: (Maximal) Valid Groups Involved in Table 3.1

The Set of Valid Groups	
$\mathbb{G}_2$	$\{u_1, u_2\}, \{u_1, u_3\}, \{u_1, u_4\}, \{u_1, u_5\}, \{u_1, u_6\}$ $\{u_2, u_3\}, \{u_2, u_4\}, \{u_2, u_5\}, \{u_2, u_6\}$ $\{u_3, u_4\}, \{u_3, u_5\}, \{u_3, u_6\}$ $\{u_4, u_5\}, \{u_4, u_6\}$ $\{u_5, u_6\}$
$\mathbb{G}_3$	$\{u_1, u_2, u_3\}, \{u_1, u_2, u_4\}, \{u_1, u_2, u_6\}, \{u_1, u_3, u_4\}, \{u_1, u_3, u_6\}, \{u_1, u_4, u_5\}$ $\{u_2, u_3, u_4\}, \{u_2, u_3, u_5\}, \{u_2, u_3, u_6\}, \{u_2, u_4, u_5\}, \{u_2, u_4, u_6\}, \{u_2, u_5, u_6\}$ $\{u_3, u_4, u_5\}, \{u_3, u_4, u_6\}, \{u_3, u_5, u_6\}$ $\{u_4, u_5, u_6\}$
$\mathbb{G}_4$	$\{u_1, u_2, u_3, u_4\}$ $\{u_2, u_3, u_4, u_5\}, \{u_2, u_3, u_4, u_6\}, \{u_2, u_3, u_5, u_6\}, \{u_2, u_4, u_5, u_6\}$ $\{u_3, u_4, u_5, u_6\}$
$\mathbb{G}_5$	$\{u_2, u_3, u_4, u_5, u_6\}$
The Set of Maximal Valid Groups	
$\mathbb{G}_M$	$\{u_1, u_2, u_6\}, \{u_1, u_3, u_6\}, \{u_1, u_4, u_5\}$ $\{u_1, u_2, u_3, u_4\}$ $\{u_2, u_3, u_4, u_5, u_6\}$

The set of maximal valid groups is denoted by $\mathbb{G}_M$. For example, in Table 5.1, among the 38 valid groups, there are only 5 maximal valid groups, as shown at the bottom. However, the 5 maximal valid groups contain the same grouping information as the 38 valid groups. Said differently, by knowing $\{u_1, u_4, u_5\}$ is a maximal valid 3-group, we can infer that $\{u_1, u_4\}$, $\{u_1, u_5\}$, and $\{u_4, u_5\}$ must be valid 2-groups based on Apriori property. Therefore, in order to discover grouping information about mobile users, we only need to find the set of maximal valid groups instead of the complete set of valid groups.

5.2 Related Work

5.2.1 Mining Maximal Frequent Itemsets

The same redundant problem also occurs in frequent itemset mining which is part of association rule mining. We therefore first examine the previous work on mining maximal frequent itemsets (MFI).

Since association rule mining was first introduced in [AIS93], there has been extensive studies on frequent itemsets mining algorithms, such as [AS94, PCY95, Toi96, HPY00]. However, all of these works attempt to find the complete set of frequent itemsets, denoted by FI, which involves a large redundant portion in most cases. That is, if a k -itemset is frequent, all $2^k - 1$ of its subsets of sizes 1 to $k - 1$ are frequent as well.

Therefore, researchers have paid attention to two alternatives including the set of *frequent closed itemsets*, denoted by FCI [PBTL99, PHM00, Zak00a, ZH02, WHP03], and the set of *maximal frequent itemsets*, denoted by MFI [Bay98, Zak00b, BCG01, KZ01]. An itemset is called *closed* if it has no superset that has the same support. A frequent itemset is called *maximal* if it is not a subset of any other frequent itemset. It is clear that $MFI \subseteq FCI \subseteq FI$. The advantage of FCI is that FCI contains the complete support information for each frequent itemset in FI and thus can be used directly to generate association rules, while the disadvantage is, in the presence of long frequent itemsets, FCI may still be a huge set. On the other hand, MFI does not contain the complete support information and thus requires one more database scan in order to compute the support counts for subsets and thus to generate association rules. But, MFI is always the smallest set comparing with FI and FCI, especially in the presence of long frequent patterns. Therefore, many researchers has focused on MFI.

Figure 5.1: An Set Enumeration Tree over Four Items

Recent methods of mining MFI normally perform a traversal on the search space which can be described by using Rymon's generic set enumeration tree framework [Rym92], as shown in Figure 5.1, where each node consists of *head* and *tail*. The head is a candidate

frequent itemset and the tail contains candidate items to form new heads of descendent nodes. For example, the node with head $\{1\}$ has tail $\{2, 3, 4\}$.

The brute force way to find MFI is to check the maximality, i.e., checking the presence of any superset, for each potential maximal frequent itemset. Initially, the search space could include all frequent itemsets in FI, which generally is a large amount. Therefore, most of the existing methods for mining MFI focus on two aspects: to reduce the search space as much as possible and to compute support count of an itemset quickly. In the first aspect, several pruning strategies have been proposed. The primary one is *superset frequency pruning*, i.e., if an itemset formed by all the items in head and tail can be determined to be frequent, the subtree under this node can be safely ignored. Note that, such a determination can be made in two ways: (1) compute the support count of this itemset; and (2) check whether this itemset can be subsumed by another maximal frequent itemset which is already found. Superset frequency pruning is first introduced in MaxMiner [Bay98], which uses both ways of frequent itemset determination. Mafia [BCG01] also adopt both ways of superset frequency pruning but in the name of FHUT and HUTMFI respectively. GenMax [KZ01] only employs the second way while the authors introduced *local maximal frequent itemsets* to accelerate the subsumption checking. MaxMiner, Mafia and GenMax also use *dynamic reordering* of items in the tail to facilitate superset frequency pruning. Other strategies for reducing search space include *subset-infrequency pruning* (MaxMiner) and *parent equivalence pruning* (Mafia).

As for fast support counting, MaxMiner tried to compute a good lower bound on the support count. Mafia used vertical bit-vector data format, and compression and projection of bitmaps. GenMax adopted *diffset propagation* method, where the diffset is defined as the difference of transaction ids for two itemsets. In addition, in terms of searching strategy, both Mafia and GenMax adopt depth-first search, while MaxMiner uses breadth-first search.

Note that, all these algorithms have a common weakness: requiring *maximality checking*, i.e., whenever a potential maximal frequent itemset is found, it is necessary to check whether it has a superset already found. This operation will become costly at the later stage of the whole mining process since the set of MFI already found is ever growing during the mining process.

5.2.2 Maximal Clique Enumeration

We have found that, based on our novel structure VG-graph introduced in Chapter 3, each maximal valid group forms a maximal *clique*. Therefore, we also studied the methods for maximal clique enumeration. Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. A *clique* in G is defined as a subset $V' \subseteq V$ of vertices, every pair of which is connected by an edge in E . In other words, a clique² is a complete subgraph of G . A clique is *maximal* if its vertices are not a subset of the vertices of a larger clique. Since the problem of enumerating all maximal cliques was first introduced in [HR57], there has been a lot of subsequent work on improving the efficiency of enumeration algorithms [AM70, MC72, BK73, Joh76, TIAS77, LT81, TTT88]. Among them, the Bron-Kerbosch algorithm [BK73] was shown to be one of the most efficient algorithms [Joh76]. The Bron-Kerbosch algorithm uses a backtracking and branch-and-bound technique to remove branches that cannot lead to maximal cliques. We will discuss Bron-Kerbosch algorithm in more details in later sections. Unfortunately, we cannot apply these maximal clique enumeration algorithms to mining maximal valid groups directly. We will see later, although a maximal valid group forms a maximal clique in VG-graph, the reverse is not true, i.e., a maximal clique in VG-graph may not represent a maximal valid group, because it may not be valid.

5.3 Extensions of AGP and VG-growth for Mining Maximal Valid Groups

5.3.1 AMG Algorithm

Our first proposed algorithm for mining maximal valid groups is **AMG** algorithm (**A**priori-based algorithm for mining **M**aximal valid **G**roups), which is derived from algorithm AGP [WLH03].

The main idea of AMG is based on the following observations. As introduced in Chapter 3, AGP can find the complete set of valid groups. Given a valid group containing $k - 1$ members, denoted by G_{k-1} , if G_{k-1} is not maximal, there must exist at least one valid k -group, denoted by G_k , such that $G_{k-1} \subset G_k$. In other words, if G_{k-1} is not a subgroup of any valid k -group, then G_{k-1} must be a maximal valid group. Formally, we have the theorem below.

²In the literatures in early years, clique was defined as the maximal complete subgraph.

Theorem 5.8 *Given $\mathbb{G}_{k-1}$ and $\mathbb{G}_k$ found by AGP algorithm, $G_{k-1} \in \mathbb{G}_{k-1}$ is a maximal valid group if $\nexists G_k \in \mathbb{G}_k$, s.t., $G_{k-1} \subset G_k$.*

Proof. This is straightforward based on the definition of maximal valid group. ■

Based on this theorem, we modify the original AGP algorithm to directly output maximal valid groups, as shown in Algorithm 5.1.

Algorithm 5.1 AMG Algorithm.

Input: D , max_dis , min_dur , and min_wei
Output: $\mathbb{G}_M$
Steps:
1: $\mathbb{G}_M = C_M = \emptyset$;
2: $\mathbb{G}_1$ = the set of all distinct users;
3: **for** ($k = 2$; $\mathbb{G}_{k-1} \neq \emptyset$; $k++$) **do**
4: *(lines 4-17 in AGP algorithm on page 33)*
5: **if** ($C_M \neq \emptyset$) **then**
6: **for** each valid k -group $G_k \in \mathbb{G}_k$ **do**
7: remove all $(k-1)$ subgroups of G_k from C_M ;
8: **end for**
9: $\mathbb{G}_M = \mathbb{G}_M \cup C_M$;
10: **end if**
11: $C_M = \mathbb{G}_k$;
12: **end for**
13: **return** $\mathbb{G}_M$;

In Algorithm 5.1, for each k , C_M first holds the set of valid $(k-1)$ -groups. After finding the set of valid k -groups, those valid $(k-1)$ -groups which has a valid k -supergroup are removed from C_M , making C_M only hold maximal valid $(k-1)$ -groups. Figure 5.2 shows the search tree for AMG. As we can see, AMG performs a breadth-first search and visits the same number of nodes as AGP and VG-growth. Although AMG traverses nodes in the same order as AGP, AMG makes level-wise pruning, i.e., removing all $k - 1$ subgroups for each node at depth k ($k > 2$). Like AGP, AMG still suffers from the overhead of candidate generation and multiple database scans, making it less efficient.

5.3.2 VGMax Algorithm

VGMax is an extension of VG-growth with some modifications to generate only maximal valid groups.

Figure 5.2: Search Tree for AMG

Before presenting VGMax, we first look at some important properties of VG-growth which are related to maximal valid groups.

Theorem 5.9 *Given conditional VG-graph $VG^k(H) = (V^k, E^k, s^k)$ of $H = \{u_{a_1}, u_{a_2}, \dots, u_{a_k}\}$ ($a_1 < a_2 < \dots < a_k$) and any edge $(u_i, u_j) \in E^k$ ($a_k < i < j$), $G = H \cup \{u_i, u_j\}$ is NOT a maximal valid group if $u_j \in V^{k+1}$, where $VG^{k+1}(H \cup \{u_i\}) = (V^{k+1}, E^{k+1}, s^{k+1})$.*

Proof. First, we know from Lemma 3.1 that G is a $(k+2)$ -valid group. Now we need to prove that, if $u_j \in V^{k+1}$, there must exist at least one valid group, say G' , s.t., $G \subset G'$.

Recall Equation Eq. 3.6. If $u_j \in V^{k+1}$, $\exists (u_j, u_l) \in E^{k+1}$. Then, based on Lemma 3.1, we know that $H' \cup \{u_j, u_l\}$ is a valid $(k+3)$ -group. Let $G' = H' \cup \{u_j, u_l\}$, and $G \subset G'$. Hence, this theorem holds. ■

This theorem says, among the set of suffix-neighbors of u_i , the ones (e.g., u_j) who appear in the higher order conditional VG-graph for $H \cup \{u_i\}$ cannot lead to a maximal valid group $H \cup \{u_i, u_j\}$. Therefore, we can make some modifications to lines 5-7 in VG-growth (see Algorithm 3.3) to avoid generating those valid groups that cannot be maximal. Instead of generating a valid group for each suffix-neighbor, we only generate a *candidate maximal*

valid group for those prefix-neighbors that do not appear in the higher order conditional VG-graph.

Definition 5.3 *Given conditional VG-graph $VG^k(H) = (V^k, E^k, s^k)$ of $H = \{u_{a_1}, u_{a_2}, \dots, u_{a_k}\}$ ($a_1 < a_2 < \dots < a_k$) and any edge $(u_i, u_j) \in E^k$ ($a_k < i < j$), $G_c = H \cup \{u_i, u_j\}$ is a **candidate maximal valid group** if $u_j \notin V^{k+1}$, where $VG^{k+1}(H \cup \{u_i\}) = (V^{k+1}, E^{k+1}, s^{k+1})$.*

For example, consider the mining process of VG-growth on vertex u_1 , as shown in Figure 5.3. The suffix-neighbors of u_1 in $VG^0(\emptyset)$ are $\{u_2, u_3, u_4, u_5, u_6\}$. They all appear in the conditional VG-graph $VG^1(\{u_1\})$. Thus, VGMax will not generate $\{u_1, u_2\}$, $\{u_1, u_3\}$, ..., $\{u_1, u_6\}$ as candidate maximal valid group. On the other hand, when visiting vertex u_2 in $VG^1(\{u_1\})$, $\{u_1, u_2, u_6\}$ will be generated as a candidate maximal valid group, since u_6 does not appear in $VG^2(\{u_1, u_2\})$.

Figure 5.3: An Illustrative Example for VGMax

However, we cannot confirm a candidate maximal valid group to be a maximal valid group until it has no other supergroup. That is, we have to perform maximality checking for each candidate maximal valid group. Fortunately, we can do this on the fly as each maximal valid group is found, instead of waiting until all candidate maximal valid groups are identified. The following two properties provide support for this.

Theorem 5.10 *Given two candidate maximal valid groups G_c and G' , if $G_c \subset G'$, then G' must be found earlier than G_c by VG-growth.*

Proof. Without loss of generality, assume $G_c = \{u_{a_1}, \dots, u_{a_m}\}$ ($a_1 < \dots < a_m$) and $G' = \{u_{a_1}, \dots, u_{a_m}, u_{a_n}\}$. We will first show that a_n must be less than a_m and then prove that G' must be generated by VG-growth earlier than G_c .

First, assume that $a_n > a_m$. There must exist an edge (u_{a_m}, u_{a_n}) in the $(m-1)$ -order conditional VG-graph $VG^{m-1}(H)$ with $H = \{u_{a_1}, u_{a_2}, \dots, u_{a_{m-1}}\}$. That is, u_{a_m} appears in $VG^{m-1}(H)$. This conflicts with the fact that G_c is a candidate maximal valid group. Thus, we have $a_n < a_m$.

Next, we rewrite G_c and G' as $G_c = \{u_{a_1}, u_{a_2}, \dots, u_{a_k}, u_{a_{k+1}}, \dots, u_{a_m}\}$ and $G' = \{u_{a_1}, u_{a_2}, \dots, u_{a_k}, u_{a_n}, u_{a_{k+1}}, \dots, u_{a_m}\}$ ($0 \leq k \leq m-1$). Consider the four nodes: $n(\{u_{a_1}, u_{a_2}, \dots, u_{a_k}, u_{a_{k+1}}\})$, $n(\{u_{a_1}, u_{a_2}, \dots, u_{a_k}, u_{a_n}\})$, $n(G_c)$, and $n(G')$. For simplicity, let n_1 and n_2 denote the first two nodes respectively. Thus, $n(G_c)$ is a descendent node of n_1 and $n(G')$ is a descendent node of n_2 . Note that, n_2 is a left sibling node of n_1 , since $a_n < a_{k+1}$. Therefore, VG-growth traverses the four nodes in order: $n_2 \rightarrow n_1 \rightarrow n(G') \rightarrow n(G_c)$. That is to say, G' is generated earlier than G_c by VG-growth. This theorem holds. ■

This theorem states that, for a candidate maximal valid group G_c , if there exist another candidate maximal valid group G' such that G_c is a subgroup of G' , then G' must be found by VG-growth earlier than G_c .

Theorem 5.11 *Let $\mathbb{G}_C$ be the set of candidate maximal valid groups, then the set of maximal valid groups $\mathbb{G}_M = \mathbb{G}'_M$, where $\mathbb{G}'_M = \{G_c \mid G_c \in \mathbb{G}_C, \text{ and } \nexists G' \in \mathbb{G}_C, \text{ s.t., } G_c \subset G' \text{ and node } n(G') \text{ is visited before } n(G_c) \text{ in search tree of VG-growth}\}$.*

Proof. We need to prove that $\mathbb{G}_M \subseteq \mathbb{G}'_M$ and $\mathbb{G}'_M \subseteq \mathbb{G}_M$.

We first prove that $\mathbb{G}_M \subseteq \mathbb{G}'_M$. This is straightforward based on the definitions of maximal valid group and candidate maximal valid group.

Next, we prove that $\mathbb{G}'_M \subseteq \mathbb{G}_M$. That is, $\forall G_c \in \mathbb{G}_C$, G_c must be a maximal valid group if $\nexists G' \in \mathbb{G}_C$, such that $G_c \subset G'$ and node $n(G')$ is visited before $n(G_c)$ in the search tree of VG-growth. We prove this by contradiction. That is, assume that such G_c is not a maximal valid group. Thus, there must exist another candidate maximal valid group G'_c such that

$G_c \subset G'_c$ and node $n(G'_c)$ is visited after $n(G_c)$ in the search tree of VG-growth. However, this contradicts with Theorem 5.10.

Hence, this theorem holds. ■

Given a candidate maximal valid group G_c and the set of maximal valid groups $\mathbb{G}_M$ found already, the above two properties ensure that G_c must be a maximal valid group if it has no supergroup in $\mathbb{G}_M$.

To summarize, we first identify the set of candidate maximal valid groups. We then perform maximality checking for each candidate maximal valid group against the set of maximal valid groups already found. The complete VGMax algorithm is shown in Algorithm 5.2, and Figure 5.4 gives the search tree of applying VGMax on the running example database. As shown in Figure 5.4, VGMax visits nodes in the same order as VG-growth. However, instead of output each node as a valid group, VGMax first determines whether a node is a candidate maximal valid group and then performs maximality checking if the answer is positive.

Figure 5.4: Search Tree for VGMax

Theorem 5.12 *Algorithm VGMax only generates maximal valid groups.*

Algorithm 5.2 VGMax Algorithm.

Input: D , max_dis , min_dur , and min_wei **Output:** $\mathbb{G}_M$ **Steps:**

- 1: $\mathbb{G}_M = \emptyset$;
- 2: use AGP to find $\mathbb{G}_2$ and $s(\mathbb{G}_2)$;
- 3: construct VG-graph $VG^0(\emptyset) = (V^0, E^0, s^0)$ based on Definition 3.6;
- 4: **VGMax**($VG^0(\emptyset)$, $\emptyset$, $\mathbb{G}_M$);
- 5: **return** $\mathbb{G}_M$;

Function **VGMax**(VG-Graph, H , $\mathbb{G}_M$) // VG-graph = (V, E, s)

- 1: **for** each vertex $u_i \in V$ **do**
- 2: $H' = H \cup \{u_i\}$;
- 3: $V_{H'} = \{u_j \mid u_j \in V, (u_j, u_i) \in E, j > i\}$;
- 4: construct conditional VG-graph $VG^{|H'|}(H') = (V^{|H'|}, E^{|H'|}, s^{|H'|})$;
- 5: **if** $E^{|H'|} \neq \emptyset$ **then**
- 6: **for** each vertex $u \in V_{H'}$ **do**
- 7: **if** $u \notin V^{|H'|}$ **then**
- 8: obtain a candidate maximal valid group: $G = H' \cup \{u\}$;
- 9: **MaximalityChecking**($G, \mathbb{G}_M$);
- 10: **end if**
- 11: **end for**
- 12: **VGMax**($VG^{|H'|}(H')$, H' , $\mathbb{G}_M$);
- 13: **else**
- 14: **for** each vertex $u \in V_{H'}$ **do**
- 15: obtain a candidate maximal valid group: $G = H' \cup \{u\}$;
- 16: **MaximalityChecking**($G, \mathbb{G}_M$);
- 17: **end for**
- 18: **end if**
- 19: **end for**

Function **MaximalityChecking**($G, \mathbb{G}_M$)

- 1: **if** $\nexists G' \in \mathbb{G}_M$, s.t., $G \subset G'$ **then**
 - 2: $\mathbb{G}_M = \mathbb{G}_M \cup \{G\}$;
 - 3: **end if**
-

Proof. This is straightforward based on Theorem 5.10. ■

Theorem 5.13 *Algorithm VGMax will generate all maximal valid groups.*

Proof. Let G_M be any maximal valid group, we need to prove that VGMax will correctly find it.

Without loss of generality, assume $G_M = \{u_{a_1}, \dots, u_{a_m}\}$ ($a_1 < \dots < a_m$). G_M will be identified as a candidate maximal valid group when visiting vertex $u_{a_{m-1}}$ in $VG^{m-2}(\{u_1, \dots, u_{a_{m-2}}\})$. And $\nexists G'$, s.t., $G_M \subset G'$, since G_M is a maximal valid group. Therefore, G_M will be identified as a maximal valid group by VGMax. ■

5.4 Mining Maximal Valid Groups by Finding Maximal Cliques

In this section, we present a VG-graph-based algorithm, namely VGBK, for mining maximal valid groups without maximality checking. That is, the maximal valid groups can be directly identified based on simple criteria, instead of checking whether it has a supergroup which is already found.

5.4.1 Bron-Kerbosch Algorithm

The main idea of VGBK algorithm comes from the Bron-Kerbosch algorithm [BK73], which is one of the most efficient algorithms for enumerating all maximal cliques in an undirected graph [Joh76]. A maximal clique is a maximal complete subgraph which is not contained by any other complete subgraph. In [BK73], C. Bron and K. Kerbosch proposed a backtracking algorithm which employs a branch-and-bound technique to prune branches that cannot lead to maximal cliques³, as shown in Algorithm 5.3.

Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Bron-Kerbosch algorithm returns all maximal cliques $\mathcal{C}_M$. There are three important sets involved in Bron-Kerbosch algorithm: *Compsub*, *Not* and *Cand*. *Compsub* is the set of vertices to be extended by a new vertex. *Not* and *Cand* contain all vertices that are eligible to extend *Compsub*, i.e., the set of vertices

³In fact, there are two versions of Bron-Kerbosch algorithm. Algorithm 5.3 only presents the first version, because the idea in the second version does not work for VG-graph. For more details of the second version, please refer to [BK73].

Algorithm 5.3 Bron-Kerbosch Algorithm (Version 1).**Input:** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: an undirected graph**Output:** $\mathcal{C}_M$: all maximal cliques**Steps:**1: $\mathcal{C}_M = \text{Compsub} = \text{Not} = \emptyset$;2: $\text{Cand} = \mathcal{V}$;3: **BKV1**($\mathcal{G}, \text{Not}, \text{Cand}, \text{Compsub}, \mathcal{C}_M$);4: **return** $\mathcal{C}_M$;**Function** **BKV1**($\mathcal{G}, \text{Not}, \text{Cand}, \text{Compsub}, \mathcal{C}_M$)1: **for** each $v_i \in \text{Cand}$ **do**2: add v_i to Compsub ;3: $\text{Not}_{\text{new}} = \{v_j \mid (v_i, v_j) \in \mathcal{E}, v_j \in \text{Not}\}$;4: $\text{Cand}_{\text{new}} = \{v_k \mid (v_i, v_k) \in \mathcal{E}, v_k \in \text{Cand}, k > i\}$;5: **if** ($\text{Not}_{\text{new}} = \emptyset$ **AND** $\text{Cand}_{\text{new}} = \emptyset$) **then**6: add Compsub to $\mathcal{C}_M$;//A maximal clique is found7: **else**8: **if** ($\text{Cand}_{\text{new}} \neq \emptyset$) **then**9: **BKV1**($\mathcal{G}, \text{Not}_{\text{new}}, \text{Cand}_{\text{new}}, \text{Compsub}, \mathcal{C}_M$);10: **end if**11: **end if**12: remove v_i from Compsub and add to Not ;13: **end for**

connected to *all* vertices in Compsub . These eligible vertices are divided into two sets: Not contains the set of vertices already added to Compsub and Cand contains those vertices to be added to Compsub at a later stage. The key operation of Bron-Kerbosch algorithm involves a recursive call to a function, namely **BKV1**, to generate all possible extensions to Compsub by adding vertices from Cand and ensuring no vertices from Not are included in the new extensions. **BKV1** function performs the following operations.

- (i) Select a vertex v_i in turn from Cand
- (ii) Add v_i into Compsub
- (iii) Generate two new sets Not_{new} and Cand_{new} from Not and Cand respectively by removing all vertices not connected to v_i in $\mathcal{G}$, keeping the old sets intact
- (iv) Recursively call **BKV1** function to operate on the new sets, if Cand_{new} is not empty
- (v) Upon return, remove v_i from Compsub and add v_i into Not

It is easy to see that $Cand = \emptyset$ is a necessary condition to identify $Compsub$ as a maximal clique. However, it is not sufficient. Because if Not is not empty, we know that $Compsub$ must be subsumed in some larger cliques formed by $Compsub$ and the vertices in Not and thus is not a maximal clique. Therefore, the criteria to identify $Compsub$ as a maximal clique is to have both $Cand$ and Not empty, see lines 5 and 6 in BKV1 function.

A basic operation of Bron-Kerbosch algorithm is the determination of whether two vertices are connected in the graph. Note that, the connection relationships between vertices are *static* and *invariant* in a normal graph. Unfortunately, that is not the case for VG-graph and conditional VG-graph. A maximal clique in VG-graph may not represent a maximal valid group. For example, in Figure 3.2, all the six vertices u_1, u_2, u_3, u_4, u_5 , and u_6 form a maximal clique. However, it does not represent a maximal valid group since $\{u_1, u_2, u_3, u_4, u_5, u_6\}$ is not a valid group as the connection relationships between the vertices in the VG-graph and conditional VG-graphs change dynamically according to different prefix-groups. That is, some vertices are connected in the original VG-graph or a particular conditional VG-graph for certain prefix-group, but they may not be connected in another conditional VG-graph for another prefix-group. For example, u_4 and u_6 are connected in the original VG-graph because $\{u_4, u_6\}$ is a valid 2-group. However, they are not connected in $VG^1(\{u_1\})$, as shown in Figure 5.3.

5.4.2 VGBK Algorithm

Fortunately, the above problems can be solved by combining conditional VG-graph with Bron-Kerbosch algorithm, leading to the VGBK algorithm. The main idea of VGBK is to maintain four important variables α , $VG^{|\alpha|}(\alpha)$, V^- , and V^+ during the mining process. α is a valid group which is to be augmented with more users, similar to the prefix-group in VG-growth. $VG^{|\alpha|}(\alpha)$ is the conditional VG-graph for prefix-group α . V^- is a list of users already added to α as suffix-neighbors to form a larger valid group. V^+ is a list of users to be added to α .

VGBK algorithm is shown in Algorithm 5.4. Similar to Bron-Kerbosch algorithm, the key operation of VGBK algorithm involves a recursive call to a function, called VGBK, to generate all possible extensions to α by adding users from V^+ and ensuring no users from V^- are included in the new extensions. In particular, VGBK function consists of the following steps.

Algorithm 5.4 VGBK Algorithm.**Input:** D , max_dis , min_dur , and min_wei **Output:** $\mathbb{G}_M$ **Steps:**

- 1: $\mathbb{G}_M = V^- = \alpha = \emptyset$;
- 2: use AGP to find $\mathbb{G}_2$ and $s(\mathbb{G}_2)$;
- 3: construct VG-graph $VG^0(\emptyset) = (V^0, E^0, s^0)$ based on Definition 3.6;
- 4: $V^+ = \{u \mid u \in V^0\}$;
- 5: **VGBK**($VG^0(\emptyset)$, V^- , V^+ , α , $\mathbb{G}_M$);
- 6: **return** $\mathbb{G}_M$;

Function VGBK(VG-graph, V^- , V^+ , α , $\mathbb{G}_M$) // VG-graph = (V, E, s)

- 1: **for** each $u_i \in V^+$ **do**
- 2: $\beta = \alpha \cup \{u_i\}$;
- 3: $V_{new}^- = \{u_j \mid (u_i, u_j) \in E, u_j \in V^-\}$;
- 4: $V_{new}^+ = \{u_k \mid (u_i, u_k) \in E, u_k \in V^+, k > i\}$;
- 5: **if** ($V_{new}^- = \emptyset$ **AND** $V_{new}^+ = \emptyset$) **then**
- 6: $\mathbb{G}_M = \mathbb{G}_M \cup \beta$; // A maximal valid group is found
- 7: **else**
- 8: **if** ($V_{new}^+ \neq \emptyset$) **then**
- 9: $V_\beta = \{u_l \mid (u_i, u_l) \in E\}$;
- 10: construct $VG^{|\beta|}(\beta)$ based on Equations Eq. 3.5 ~ Eq. 3.9 but using V_β ;
- 11: **VGBK**($VG^{|\beta|}(\beta)$, V_{new}^- , V_{new}^+ , β , $\mathbb{G}_M$);
- 12: **end if**
- 13: **end if**
- 14: $V^- = V^- \cup \{u_i\}$;
- 15: **end for**

- (i) Select a user u_i from V^+
- (ii) Add u_i into α
- (iii) Generate two new lists V_{new}^- and V_{new}^+ from V^- and V^+ respectively by choosing those users connected to u_i in $VG^{|\alpha|}(\alpha)$
- (iv) If V_{new}^+ is not empty, construct the conditional VG-graph for prefix-group $\alpha \cup \{u_i\}$
- (v) Recursively call VGBK function
- (vi) Upon return, remove u_i from α and add u_i into V^-

Similarly, the criteria to identify α as a maximal valid group is *both* V^- and V^+ are empty.

Theorem 5.14 In VGBK algorithm, α is a maximal valid group iff $V^+ = \emptyset$ and $V^- = \emptyset$.

Proof. This is straightforward based on the definitions of V^- and V^+ . ■

Figure 5.5: Search Tree for VGBK

Figure 5.5 shows the search tree for VGBK. Note that, there are two differences between VGBK and VGMax. Firstly, VGMax performs a depth-first search on the search tree, while VGBK performs a pre-order traversal.

The second difference lies in the fact that VGBK maintains the list V^- to keep track of the vertices that already served as a suffix to α . With this, VGBK avoids maximality checking completely. However, in order to correctly update list V^- , we need to construct larger conditional VG-graphs using all neighbors instead of only the suffix-neighbors in order to update V^- correctly (refer to lines 9 ~ 10 in VGBK function).

Consider the example of applying VGBK on the VG-graph in Figure 3.2. Table 5.2 illustrates the whole mining process by showing the status of each variable for every iteration of lines 2-13 in Function VGBK ⁴ (see Algorithm 5.4).

Note that, to make a comparison, we also show the corresponding conditional VG-graphs used by VG-growth at the right hand side in Figure 5.6. It is clear that VGBK construct

⁴We only give the user id in Table 5.2, making it easy to look across this table and Figure 5.5

Table 5.2: Complete Mining Process of VGBK on Figure 3.2

VG-graph	α	V^-	V^+	u_i	β	V_{new}^-	V_{new}^+	$VG^{ \beta }(\beta)$
$VG^0(\emptyset)$	$\emptyset$	$\emptyset$	$\{1, 2, 3, 4, 5, 6\}$	1	$\{1\}$	$\emptyset$	$\{2, 3, 4, 5, 6\}$	$VG^1(\{1\})$
$VG^1(\{1\})$	$\{1\}$	$\emptyset$	$\{2, 3, 4, 5, 6\}$	2	$\{1, 2\}$	$\emptyset$	$\{3, 4, 6\}$	$VG^2(\{1, 2\})$
$VG^2(\{1, 2\})$	$\{1, 2\}$	$\emptyset$	$\{3, 4, 6\}$	3	$\{1, 2, 3\}$	$\emptyset$	$\{4\}$	$VG^3(\{1, 2, 3\})$
$VG^3(\{1, 2, 3\})$	$\{1, 2, 3\}$	$\emptyset$	$\{4\}$	4	$\{1, 2, 3, 4\}$	$\emptyset$	$\emptyset$	
$VG^2(\{1, 2\})$	$\{1, 2\}$	$\{3\}$	$\{3, 4, 6\}$	4	$\{1, 2, 4\}$	$\{3\}$	$\emptyset$	
$VG^2(\{1, 2\})$	$\{1, 2\}$	$\{3, 4\}$	$\{3, 4, 6\}$	6	$\{1, 2, 6\}$	$\emptyset$	$\emptyset$	
$VG^1(\{1\})$	$\{1\}$	$\{2\}$	$\{2, 3, 4, 5, 6\}$	3	$\{1, 3\}$	$\{2\}$	$\{4, 6\}$	$VG^2(\{1, 3\})$
$VG^2(\{1, 3\})$	$\{1, 3\}$	$\{2\}$	$\{4, 6\}$	4	$\{1, 3, 4\}$	$\{2\}$	$\emptyset$	
$VG^2(\{1, 3\})$	$\{1, 3\}$	$\{2, 4\}$	$\{4, 6\}$	6	$\{1, 3, 6\}$	$\emptyset$	$\emptyset$	
$VG^1(\{1\})$	$\{1\}$	$\{2, 3\}$	$\{2, 3, 4, 5, 6\}$	4	$\{1, 4\}$	$\{2, 3\}$	$\{5\}$	$VG^2(\{1, 4\})$
$VG^2(\{1, 4\})$	$\{1, 4\}$	$\{2, 3\}$	$\{5\}$	5	$\{1, 4, 5\}$	$\emptyset$	$\emptyset$	
$VG^1(\{1\})$	$\{1\}$	$\{2, 3, 4\}$	$\{2, 3, 4, 5, 6\}$	5	$\{1, 5\}$	$\{4\}$	$\emptyset$	
$VG^1(\{1\})$	$\{1\}$	$\{2, 3, 4, 5\}$	$\{2, 3, 4, 5, 6\}$	6	$\{1, 6\}$	$\{2, 3\}$	$\emptyset$	
$VG^0(\emptyset)$	$\emptyset$	$\{1\}$	$\{1, 2, 3, 4, 5, 6\}$	2	$\{2\}$	$\{1\}$	$\{3, 4, 5, 6\}$	$VG^1(\{2\})$
$VG^1(\{2\})$	$\{2\}$	$\{1\}$	$\{3, 4, 5, 6\}$	3	$\{2, 3\}$	$\{1\}$	$\{4, 5, 6\}$	$VG^2(\{2, 3\})$
$VG^2(\{2, 3\})$	$\{2, 3\}$	$\{1\}$	$\{4, 5, 6\}$	4	$\{2, 3, 4\}$	$\{1\}$	$\{5, 6\}$	$VG^3(\{2, 3, 4\})$
$VG^3(\{2, 3, 4\})$	$\{2, 3, 4\}$	$\{1\}$	$\{5, 6\}$	5	$\{2, 3, 4, 5\}$	$\emptyset$	$\{6\}$	$VG^4(\{2, 3, 4, 5\})$
$VG^4(\{2, 3, 4, 5\})$	$\{2, 3, 4, 5\}$	$\emptyset$	$\{6\}$	6	$\{2, 3, 4, 5, 6\}$	$\emptyset$	$\emptyset$	
$VG^3(\{2, 3, 4\})$	$\{2, 3, 4\}$	$\{1, 5\}$	$\{5, 6\}$	6	$\{2, 3, 4, 6\}$	$\{5\}$	$\emptyset$	
$VG^2(\{2, 3\})$	$\{2, 3\}$	$\{1, 4\}$	$\{4, 5, 6\}$	5	$\{2, 3, 5\}$	$\{4\}$	$\{6\}$	$VG^3(\{2, 3, 5\})$
$VG^3(\{2, 3, 5\})$	$\{2, 3, 5\}$	$\{4\}$	$\{6\}$	6	$\{2, 3, 5, 6\}$	$\{4\}$	$\emptyset$	
$VG^2(\{2, 3\})$	$\{2, 3\}$	$\{1, 4, 5\}$	$\{4, 5, 6\}$	6	$\{2, 3, 6\}$	$\{4, 5\}$	$\emptyset$	
$VG^1(\{2\})$	$\{2\}$	$\{1, 3\}$	$\{3, 4, 5, 6\}$	4	$\{2, 4\}$	$\{1, 3\}$	$\{5, 6\}$	$VG^2(\{2, 4\})$
$VG^2(\{2, 4\})$	$\{2, 4\}$	$\{1, 3\}$	$\{5, 6\}$	5	$\{2, 4, 5\}$	$\{3\}$	$\{6\}$	$VG^3(\{2, 4, 5\})$
$VG^3(\{2, 4, 5\})$	$\{2, 4, 5\}$	$\{3\}$	$\{6\}$	6	$\{2, 4, 5, 6\}$	$\{3\}$	$\emptyset$	
$VG^2(\{2, 4\})$	$\{2, 4\}$	$\{1, 3, 5\}$	$\{5, 6\}$	6	$\{2, 4, 6\}$	$\{3, 5\}$	$\emptyset$	
$VG^1(\{2\})$	$\{2\}$	$\{1, 3, 4\}$	$\{3, 4, 5, 6\}$	5	$\{2, 5\}$	$\{3, 4\}$	$\{6\}$	$VG^2(\{2, 5\})$
$VG^2(\{2, 5\})$	$\{2, 5\}$	$\{3, 4\}$	$\{6\}$	6	$\{2, 5, 6\}$	$\{3, 4\}$	$\emptyset$	
$VG^1(\{2\})$	$\{2\}$	$\{1, 3, 4, 5\}$	$\{3, 4, 5, 6\}$	6	$\{2, 6\}$	$\{1, 3, 4, 5\}$	$\emptyset$	
$VG^0(\emptyset)$	$\emptyset$	$\{1, 2\}$	$\{1, 2, 3, 4, 5, 6\}$	3	$\{3\}$	$\{1, 2\}$	$\{4, 5, 6\}$	$VG^1(\{3\})$
$VG^1(\{3\})$	$\{3\}$	$\{1, 2\}$	$\{4, 5, 6\}$	4	$\{3, 4\}$	$\{1, 2\}$	$\{5, 6\}$	$VG^2(\{3, 4\})$
$VG^2(\{3, 4\})$	$\{3, 4\}$	$\{1, 2\}$	$\{5, 6\}$	5	$\{3, 4, 5\}$	$\{2\}$	$\{6\}$	$VG^3(\{3, 4, 5\})$
$VG^3(\{3, 4, 5\})$	$\{3, 4, 5\}$	$\{2\}$	$\{6\}$	6	$\{3, 4, 5, 6\}$	$\{2\}$	$\emptyset$	
$VG^2(\{3, 4\})$	$\{3, 4\}$	$\{1, 2, 5\}$	$\{5, 6\}$	6	$\{3, 4, 6\}$	$\{2, 5\}$	$\emptyset$	
$VG^1(\{3\})$	$\{3\}$	$\{1, 2, 4\}$	$\{4, 5, 6\}$	5	$\{3, 5\}$	$\{2, 4\}$	$\{6\}$	$VG^2(\{3, 5\})$
$VG^2(\{3, 5\})$	$\{3, 5\}$	$\{2, 4\}$	$\{6\}$	6	$\{3, 5, 6\}$	$\{2, 4\}$	$\emptyset$	
$VG^1(\{3\})$	$\{3\}$	$\{1, 2, 4, 5\}$	$\{4, 5, 6\}$	6	$\{3, 6\}$	$\{1, 2, 4, 5\}$	$\emptyset$	
$VG^0(\emptyset)$	$\emptyset$	$\{1, 2, 3\}$	$\{1, 2, 3, 4, 5, 6\}$	4	$\{4\}$	$\{1, 2, 3\}$	$\{5, 6\}$	$VG^1(\{4\})$
$VG^1(\{4\})$	$\{4\}$	$\{1, 2, 3\}$	$\{5, 6\}$	5	$\{4, 5\}$	$\{1, 2, 3\}$	$\{6\}$	$VG^2(\{4, 5\})$
$VG^2(\{4, 5\})$	$\{4, 5\}$	$\{1, 2, 3\}$	$\{6\}$	6	$\{4, 5, 6\}$	$\{2, 3\}$	$\emptyset$	
$VG^1(\{4\})$	$\{4\}$	$\{1, 2, 3, 5\}$	$\{5, 6\}$	6	$\{4, 6\}$	$\{2, 3, 5\}$	$\emptyset$	
$VG^0(\emptyset)$	$\emptyset$	$\{1, 2, 3, 4\}$	$\{1, 2, 3, 4, 5, 6\}$	5	$\{5\}$	$\{1, 2, 3, 4\}$	$\{6\}$	$VG^1(\{5\})$
$VG^1(\{5\})$	$\{5\}$	$\{1, 2, 3, 4\}$	$\{6\}$	6	$\{5, 6\}$	$\{2, 3, 4\}$	$\emptyset$	
$VG^0(\emptyset)$	$\emptyset$	$\{1, 2, 3, 4, 5\}$	$\{1, 2, 3, 4, 5, 6\}$	6	$\{6\}$	$\{1, 2, 3, 4, 5\}$	$\emptyset$

Figure 5.6: Different Conditional VG-graphs used by VGBK and VG-growth

the conditional VG-graph by selecting all neighbors while VG-growth only selects the suffix-neighbors. For example, when constructing $VG^1(\{u_3\})$ from the original VG-graph, VGBK uses all five neighbors of u_3 , i.e., u_1, u_2, u_4, u_5 , and u_6 . In contrast, VG-growth only selects the suffix-neighbors of u_3 , i.e., u_4, u_5, u_6 , leading to a smaller conditional VG-graph. Such an extra overhead in VGBK is necessary to track V^- . For example, consider the call to VGBK function where

$$VG-graph = VG^1(\{u_3\}) \quad \alpha = \{u_3\} \quad V^- = \{u_1, u_2\} \quad V^+ = \{u_4, u_5, u_6\}$$

From V^+ , we select $u_i = u_4$ and thus $\beta = \{u_3, u_4\}$. Then, create new sets of V^- and V^+ based on the connections of u_4 in $VG^1(\{u_3\})$. Because u_1 and u_2 are connected with u_4 in

$VG^1(\{u_3\})$, we have $V_{new}^- = \{u_1, u_2\}$. Similarly, both u_5 and u_6 are neighbors of u_4 , leading to $V_{new}^+ = \{u_5, u_6\}$. Since V_{new}^+ is not empty, we continue to construct $VG^2(\{u_3, u_4\})$ and move to the next iteration.

Theorem 5.15 *Algorithm VGBK only generates maximal valid groups.*

Proof. This is straightforward based on Theorem 5.14. ■

Theorem 5.16 *Algorithm VGBK will generates all maximal valid groups.*

Proof. Let G_M be any maximal valid group, we need to prove that VGBK will correctly identify it.

Note that, although the traversing order of VGBK is different from VGMax, VGBK still traverses the same set of nodes as VGMax. Consider the node with $\alpha = G_M$, both V^+ and V^- must be empty, since G_M is a maximal valid group. Therefore, G_M will be identified as a maximal valid group by VGBK. ■

5.5 Performance Evaluation

In this section, we evaluate the performance of AMG, VGMax, and VGBK.

5.5.1 Experimental Design and Data Generation

The proposed algorithms were implemented in C++ in Windows environment. A Pentium-IV machine with a CPU clock rate of 2.4 GHz and 1 GB of main memory was used to conduct the experiments. We used movement databases generated by IBM City Simulator [KMJ01]. City Simulator can generate realistic three-dimensional user movement over city layout that includes streets and buildings. In our experiments, the simulated user movement area covers a $1000m \times 1500m \times 100m$ space including 48 roads and 72 buildings with different heights (the highest is around 90 meters). Two datasets were generated. One is a smaller dataset containing 1000 users, while the other is a larger dataset containing 6000 users. There are 1000 time points in both datasets. In the experiments, the unit of distance measure is meter, and the interval between every two consecutive time points represents 10 minutes. Thus, $N = 1000$ represents about one week.

For each dataset, we ran the proposed algorithms with different thresholds and recorded the execution times. In particular, for each run, we fixed two of the three thresholds max_dis , min_dur and min_wei , and changed the value of the remaining one. The parameter settings are shown in Table 5.3. The criteria of choosing different parameters is to generate proper number of maximal valid groups so as to evaluate the performance of the three algorithms.

Table 5.3: Experimental Settings

Dataset	Algorithm	M	N	max_dis	min_dur	min_wei
M1kN1k	AMG VGMax VGBK	1000	1000	10 ~ 50	4	10%
				50	4 ~ 20	5%
				50	4	3 ~ 20(%)
M6kN1k		6000	1000	5 ~ 20	4	10%
				10	4 ~ 20	10%
				10	4	3 ~ 10(%)

Before presenting the experimental results, it is important to note that all of the proposed algorithms start from a same step: finding the set of valid 2-groups. The differences between the four algorithms only occur after this step. In fact, the total execution time, denoted by T , can be divided into two components: the time used to find valid 2-groups and the time used to find maximal valid groups, denoted by T_2 and T_M respectively, as shown in Figure 5.7. Since the three algorithms have the same T_2 , we therefore compare their performance based on T_M only.

For mining $\mathbb{G}_2$, we adopted a location summarization based AGP algorithm to keep T_2 small [WLH04]. The basic idea is to reduce the number of time points to be scanned by summarizing the locations within a time window into a sphere. This summarization technique, known as SLS, reduces the number of candidate 2-groups to be examined by pre-computing the upper bounds of weight-counts for each candidate 2-group.

In the following, we compare the performance of the three algorithms based on T_M .

5.5.2 Performance Comparison

Results on M1kN1k Dataset

Figure 5.8 gives the experimental results for dataset M1kN1k with different min_wei thresholds. Figure 5.8(a) shows the movements of T_M curves for the three algorithms when

Figure 5.7: Execution Time Components

min_wei threshold changes. T_M decreases when min_wei increases due to smaller number of valid groups ($|G|$) and maximal valid groups ($|G_M|$), as shown in Figure 5.8(c). VGMMax outperforms the rest with AMG being the worst. VGBK runs slightly slower than VGMMax while AMG runs around an order of magnitude slower than the rest because it performs a breadth-first search and suffers from the generation of large number of candidate groups.

Note that, the time cost of VGMMax is mainly contributed by two parts: the cost of constructing conditional VG-graph, denoted by $T(VGMMax-conVG)$, and the cost of checking maximality, denoted by $T(VGMMax-ChkMax)$. On the other hand, the time cost of VGBK is dominated by the cost of constructing conditional VG-graph using all neighbors, denoted by $T(VGBK-conVG)$. When min_wei decreases, $|G_M|$ increases, leading to larger $T(VGMMax-conVG)$ and $T(VGBK-conVG)$, as shown in Figure 5.8(b). However, they increase with different rate. In fact, $T(VGMMax-conVG)$ increases moderately with decreasing min_wei because VGMMax only selects the suffix-neighbors to build the conditional VG-graph. In contrast, $T(VGBK-conVG)$ increases around 10 times faster than $T(VGMMax-conVG)$ due to the fact that all neighbors contribute to the construction of conditional VG-graph. As for $T(VGMMax-ChkMax)$, it increases with decreasing min_wei since

Figure 5.8: Results: Dataset M1kN1k (min_wei changes)

the number of maximal valid groups increases when min_wei decreases, as shown in Figure 5.8(c). Note that, when min_wei decreases from 5% to 3%, $T(VGMax-ChkMax)$ overtakes $T(VGBK-conVG)$, this is why VGBK runs a little faster than VGMax at $min_wei = 3\%$ (see Figure 5.8(a)).

Figure 5.9 gives the experimental results for dataset M1kN1k when max_dis or min_dur changes. We can find the similar observations regarding the performances of the three algorithms.

Figure 5.9(a) shows the T_M curves for the three algorithms when max_dis threshold changes. All T_M curves move up when max_dis increases. The reason is, with a larger max_dis threshold, there are more maximal valid groups of larger sizes, as shown in Figure

Figure 5.9: Results: Dataset M1kN1k (max_dis or min_dur changes)

5.9(b). Again, we find that VGMax runs a little faster than VGBK while AMG is the slowest. The gaps between AMG and the other two algorithms become more and more significant when max_dis increases. For a larger max_dis (e.g., > 50), AMG runs around an order of magnitude slower than the other two.

Figure 5.9(c) gives the results when min_dur threshold changes. When min_dur increases, T_M of all algorithms decrease because $|G_M|$ decreases, as shown in Figure 5.9(d). We still find that AMG is the slowest and VGMax is the fastest. For a smaller min_dur , say 4, VGMax is around 10 times faster than AMG and around 2 times faster than VGBK.

In summary, VGMax exhibits the best performance when the number of maximal valid groups is not too large. However, when $|G_M|$ increases, the time cost of maximality checking

increases significantly, making VGMax less efficient, as we will see from the results on dataset M6kN1k.

Results on M6kN1k Dataset

Figure 5.10 gives the experimental results for dataset M6kN1k. Note that, the curves for AMG are not reported here in order to clearly distinguish the curves between VGMax and VGBK. In fact, AMG is still an order of magnitude slower than VGMax and VGBK.

Figure 5.10: Results: Dataset M6kN1k

Different from dataset M1kN1k, VGBK is the best for dataset M6kN1k, as shown in Figure 5.10(a). It is around three times faster than VGMax for small min_wei . This is due to the fact that $T(\text{VGBK-conVG})$ increases slower than $T(\text{VGMax-ChkMax})$ when

min_wei decreases, or more precisely, when $|\mathbb{G}_M|$ increases, as shown in Figures 5.10(b) and (c). This is the benefit by constructing larger conditional VG-graphs to avoid maximality checking, which becomes very expensive with large number of maximal valid groups.

Another observation from Figure 5.10(c) is that, in dataset M6kN1k, the gap between $|\mathbb{G}|$ and $|\mathbb{G}_M|$ is significantly larger than those in dataset M1kN1k, as shown in Figure 5.8(c). This can be explained by the different distributions of $\mathbb{G}_M$ in the two datasets, as shown in Figures 5.8(d) and 5.10(d). For both datasets, The distributions have the similar trends when max_dis or min_dur changes. From Figure 5.8(d), we can see that the maximal group size is 5 and the majority of maximal valid groups are of size 2 or 3, which means most valid groups are also maximal valid groups. On the other hand, in dataset M6kN1k, the largest group size is 12 and the dominant maximal valid groups are of sizes $5 \sim 6$, as shown in Figure 5.10(d). So, for each maximal valid group, say, of size 5, there exist $26 (2^5 - 5 - 1)$ valid groups accordingly. In other words, the search tree for dataset M6kN1k is relatively deeper than that for dataset M1kN1k. A deeper search tree means VGBK will have to include more neighbors to build conditional VG-graph, leading to larger $T(VGBK-conVG)$. This is why, in Figure 5.10(b), $T(VGBK-conVG)$ is much larger than $T(VGMax-conVG)$ when compared to Figure 5.8(b).

Here we only present the results for different min_wei , because the same observations were found for different max_dis and min_dur . That is, $|\mathbb{G}|$ and $|\mathbb{G}_M|$ increase with increasing max_dis , or decreasing min_dur . VGBK is about $2 \sim 3$ times faster than VGMax, both of which is around an order of magnitude faster than AMG. In summary, VGBK is more efficient than AMG and VGMax if there exist large number of maximal valid groups.

5.6 Summary

This chapter reports several algorithms to mine maximal valid groups from user movement data. We formally define the maximal valid group mining problem and extend our previous valid group mining algorithms AGP and VG-growth to mine maximal valid groups. These two new algorithms are AMG and VGMax. We further design another algorithm VGBK from the Bron-Kerbosch algorithm, an efficient algorithms for enumerating all maximal cliques in an undirected graph. VGBK uses simple criteria to directly identify maximal valid group without expensive maximality checking operation. Our experimental results have shown that VGMax and VGBK are around an order of magnitude faster than AMG

in most cases. For small dataset, VGMax performs better than VGBK. However, when the number of maximal valid groups increases, the performance of VGMax degrades and eventually becomes worse than that of VGBK.

Chapter 6

Conclusion and Future Work

6.1 Summary of the Thesis

This thesis introduces group pattern mining as a new data mining problem deriving knowledge from the movement data of mobile users. The definition of group pattern is unique as it involves both time and distance thresholds to determine the relationships among users. In order to measure the interestingness of group patterns, we define the *weight* of a group pattern and introduce the *min_wei* threshold to select the set of *valid group patterns*.

This research resulted in two algorithms AGP and VG-growth for mining all valid group patterns, and their extensions for mining maximal valid group patterns. Further investigation to use location summary to reduce mining overheads was also conducted. Throughout the research, extensive experiments have been conducted to evaluate the performance of our proposed algorithms, using synthetic datasets generated by IBM City Simulator [KMJ01]. The followings are the main findings derived:

- VG-growth performs better than AGP in most cases. The reason lies in the fact that they use different mining principles. AGP is designed based on the Apriori property of valid groups and adopts a level-wise breadth-first strategy to traverse the search space, generating candidate groups at each level and scanning the movement database to filter out the invalid candidates. If the number of users and the log duration are large, the candidate generation and multiple scans on database can be very costly. On the other hand, VG-growth is designed based on a novel data structure VG-graph, which contains complete but compact information with respect to all valid groups. VG-growth traverses the search space in a depth-first manner and transforms

the problem of finding valid groups into recursive construction of conditional VG-graphs and concatenations of smaller groups. The key operation of VG-growth is the intersection of valid segments to compute the weight, which is normally more efficient than database scans.

- The time used to find valid group patterns of size two occupies a significant portion of the total running time, for both AGP and VG-growth. This is because both algorithms share the same step of finding valid 2-groups, i.e, generate all possible user pairs and then scan the database to select the valid ones. Such an operation can be very expensive if both the number of users and the log duration are large. To reduce this overhead, several location summarization based algorithms have been proposed, which share a common two-step mining framework. The proposed algorithms can reduce the number of time points to be scanned by dividing the database into equal length segments and then summarizing the locations within each segment into a summarization model. More importantly, a large portion of candidate 2-groups can be pruned by pre-computing some relevant upper bounds. The experimental results have shown that the location summarization based algorithms can significantly reduce the mining overhead for group patterns of size two. The cuboid-based methods achieve better performance than the sphere-based methods because the former normally provide more accurate location summarization even with the same summarized database size. Location summarization based algorithms, together with VG-growth, provide a complete solution for mining all valid groups.
- The complete set of valid groups contains redundant grouping information, i.e., all subgroups of a valid group are also valid. This makes the complete set of valid groups a very huge set in most cases due to the exponential enumeration nature. Such observation motivates our research on mining maximal valid groups, which are defined as those valid groups that have no supergroups. Three algorithms have been proposed including AMG, VGMax and VGBK. The first two are extensions of AGP and VG-growth respectively, with some modifications in order to identify maximal valid groups. VGBK is derived from the famous Bron-Kerbosch algorithm [BK73], which is efficient for enumerating all maximal cliques in an undirected graph. VGBK uses simple criteria to identify maximal valid group without checking maximality for

each candidate, which is a costly operation when the number of maximal valid groups are large. The experimental results have shown that VGMax and VGBK are around an order of magnitude faster than AMG in most cases. For small dataset, VGMax performs better than VGBK. However, when the number of maximal valid groups increases, the performance of VGMax degrades and eventually becomes worse than that of VGBK.

6.2 Future Work

Group pattern mining is a very new research topic. The work reported in this thesis is in fact our first step towards applying data mining techniques to movement data. This suggests much more research awaits to be conducted. We briefly discuss some of the possible research directions below.

Incorporation of multiple thresholds

Our proposed group pattern mining algorithms adopt a single configuration of *max_dis*, *min_dur* and *min_wei* thresholds throughout their execution. Future work may consider adopting multiple thresholds. For example, it will be appropriate to use a larger *max_dis* threshold for rural areas, where the population density is relatively small, and use a smaller *max_dis* for downtown areas to identify group patterns more accurately. That is, different *max_dis* thresholds may be used for different zones¹. However, incorporation of multiple thresholds will require the current group pattern definitions and mining algorithms to be changed.

First of all, a new closeness definition for a pair of users will be needed. For example, given a pair of users at a particular time point, we may derive a suitable *max_dis* threshold based on the zones where the users stay. If they are in the same zone, we can just use the *max_dis* threshold pre-specified for this zone. However, a method to derive a *max_dis* threshold will be needed for the users in different zones. For a set of user, the closeness definition also will have to be extended due to multiple *max_dis* thresholds. These two fundamental changes in definitions will lead to new definitions of valid segment and group pattern accordingly, as well as new algorithms for mining valid groups and maximal valid groups.

¹Assume that the zones are squares with equal side length and the *max_dis* thresholds are much less than the side length.

More flexible location summarization methods

In this thesis, several location summarization methods have been proposed, including SLS, CLS, GSLS, and GCLS, etc.. Note that, a common weakness of these methods is the adoption of a single time window size and a single location summarization model. That is, the movement databases of different users are divided into multiple segments with equal length. And the locations within each segment are summarized into a single summarization model.

However, this may not be the suitable method to summarize the movements of users in real life. Intuitively, people are not likely to move all the time. Instead, the common movement behavior would be like, for example, *stay in office for 4 hours → go for lunch and stay in restaurant for 40 minutes → drive a car on the highway for 30 minutes → meet a client for 2 hours*, etc. [BD02]. By using multiple time window sizes for each mobile user, the location summarization may result in smaller summarized database, while retaining the summarization accuracy. In addition, multiple summarization models may be adopted based on the movement characteristics. For example, if the distance of horizontal movement is much larger than that of vertical movement, a cuboid-based location summarization method should be preferred to achieve more accurate location summarization. On the other hand, a sphere-based method should be considered if there is no much difference between the movement distances in the three dimensions, leading to smaller summarized database.

However, the adoption of multiple time window sizes and multiple summarization models will require modifications to the existing mining framework. Multiple time window sizes may cause unsynchronized time points in the summarized databases across different users. On the other hand, to adopt multiple location summarization models will result in different instances of SM within each time window. These two changes will make the determination of closeness between two instances of SM much more complex. We shall investigate the efficient ways to cope with these changes.

Pruning methods to reduce the search space

In the studies on maximal frequent itemset mining, several pruning strategies have been commonly used for reducing the search space. It will be interesting to investigate whether these pruning strategies can be adopted for mining maximal valid groups. For example, the most commonly used pruning strategy is *superset frequency pruning*, i.e., if the itemset

formed by all the items in head and tail is frequent, there is no need to expand the subtree under this node. This can also be applied to VGMax and VGBK. In VGMax, the prefix-group is the head and the vertices in the corresponding conditional VG-graph form the tail, while, in VGBK, the head and tail are formed by α and V^+ respectively. For example, consider Figures 5.4 and 5.5. There is a node with head (3) at depth 1 in both search trees, with tail (4, 5, 6) shown implicitly. The whole subtree under this node can be safely ignored, because $\{u_3, u_4, u_5, u_6\}$ is a subgroup of $\{u_2, u_3, u_4, u_5, u_6\}$, which is found earlier. Therefore, the cost of constructing conditional VG-graphs for this subtree is saved. However, this incurs extra cost by checking whether there is a supergroup found already, especially when the number of maximal valid groups are large. The tradeoff between the benefit of the pruning strategies and the extra overhead will be our main concern in the future research.

Besides, we would like to try more different distributions to evaluate the performances of VGMax and VGBK algorithms.

Other possible extensions

Besides the above three research directions, we will also consider the following possible extensions.

- Incorporation of Location-based Services (LBS) and Geographic Information System (GIS). Location-based Services are services that exploit knowledge about where an mobile user is located. Geographic Information System is a computer system for collecting, storing, analyzing and displaying data related to positions on the earth's surface, which is often used to handle digital maps. By combining with the location information, i.e., what kind of place a physical coordinate (x, y, z) represents, one can find out not only all the members belonging to a group but also the places where this group of members often visit. A more detailed knowledge regarding the group activities can therefore be obtained, which may lay the foundation to provide location-based group specific services. For example, cinemas may consider offer group discount to those movie fans who watch movies together frequently.
- Other database transformation methods. Recall that, in Section 3.1, we mentioned that transforming the movement database to transaction database is one of the obstacles for applying the classical frequent itemset mining algorithms on group pattern

mining. We have given one possible transformation method, which will lead to extremely large number of transactions, especially for a large population. However, there may exist better ways to do such database transformation. We will consider this issue further in our future work.

- Group pattern mining on real data sets. In this thesis, we use synthetic data sets generated by IBM City Simulator [KMJ01] as we have no access to real data. However, it would make our work more convincing if we can conduct experiments on real life movement data sets.
- Use discovered group patterns in real life applications. In this thesis, we have introduced a novel method, i.e., group pattern mining, dealing with spatial temporal movement data. We believe such kind of data exist popularly in our real life. This thesis does not cover the usefulness issues, i.e., how to use the group patterns discovered in real life applications, which will be one of our future works. Intuitively, different groups may be considered *useful* only in different application contexts. For example, one group may be considered useful when you deal with the notion of “prediction” and “classification”. Another group may be useful for the notion of finding all potential groups.

Appendix A

GSLs and GCLS Methods with Maximum Speed Constraint

GSLs2 and GCLS2 are the extensions to GSLs and GCLS that incorporate *the maximum speed constraint on the mobile users*. The maximum distance a user can move within a time window w is denoted by d_w .

For GSLs2, we use d_w to determine a suitable radius scale unit γ . A smaller γ can make the radius of the derived non-grid sphere closer to the original one. Except for the computation of γ , the SM and minimum and maximum distances computations of GSLs2 are the same as GSLs.

Similarly, in GCLS2, by knowing d_w , we can construct a cube with side length equal to d_w . Then, we divide this cube into a set of *sub-cells* with sub-cell length l_{sub} , and give each sub-cell a unique *sub-cell id*. Similar to Equations Eq. 4.14 and Eq. 4.15, the computation of sub-cell id and the Reverse function can be carried out based on l_{sub} , $N_{x_{sub}}$ and $N_{y_{sub}}$, where $N_{x_{sub}} = N_{y_{sub}} = \frac{d_w}{l_{sub}}$.

As shown in Figure A.1, the dashed lines represent the sub-cells. Let the cuboid with thickened borders and two corners, v_{min} and v_{max} , be the summarized cuboid in CLS. Recall that, in GCLS, the two corners are represented by cells containing v_{min} and v_{max} . In GCLS2, the corner v_{min} is still represented by the corresponding cell containing it. However, the other corner v_{max} is represented by the sub-cell containing it, denoted by sid^{max} . That is, in GCLS2, the SM is in the form of $\langle id^{min}, sid^{max} \rangle$. For user u_i at t' , the corresponding

id^{min} and sid^{max} can be determined by:

$$id^{min} = \lfloor \frac{u_i[t'].x_{min}}{l} \rfloor + N_x \cdot \lfloor \frac{u_i[t'].y_{min}}{l} \rfloor + (N_x \cdot N_y) \cdot \lfloor \frac{u_i[t'].z_{min}}{l} \rfloor \quad (\text{Eq. A.1})$$

$$\begin{aligned} sid^{max} = & \lfloor \frac{u_i[t'].x_{max} - \lfloor \frac{u_i[t'].x_{min}}{l} \rfloor \cdot l}{l_{sub}} \rfloor \\ & + N_{x_{sub}} \cdot \lfloor \frac{u_i[t'].y_{max} - \lfloor \frac{u_i[t'].y_{min}}{l} \rfloor \cdot l}{l_{sub}} \rfloor \\ & + (N_{x_{sub}} \cdot N_{y_{sub}}) \cdot \lfloor \frac{u_i[t'].z_{max} - \lfloor \frac{u_i[t'].z_{min}}{l} \rfloor \cdot l}{l_{sub}} \rfloor \end{aligned} \quad (\text{Eq. A.2})$$

Figure A.1: Illustration of GCLS2 Methods

Similar to GCLS, in GCLS2, given two grid-spheres $\langle id_i^{min}, sid_i^{max} \rangle$ and $\langle id_j^{min}, sid_j^{max} \rangle$, the minimum and maximum distances can be determined by first deriving two larger non-grid cuboids, $(v'_{i_{min}}, v'_{i_{max}})$ and $(v'_{j_{min}}, v'_{j_{max}})$, containing the two original grid-cuboids,

i.e.,

$$\begin{aligned}
v'_{i_{min}} &= (id_i^{min}.x' \cdot l, id_i^{min}.y' \cdot l, id_i^{min}.z' \cdot l) \\
v'_{i_{max}} &= (sid_i^{max}.x' \cdot l_{sub} + l_{sub} + id_i^{min}.x' \cdot l, \\
&\quad sid_i^{max}.y' \cdot l_{sub} + l_{sub} + id_i^{min}.y' \cdot l, \\
&\quad sid_i^{max}.z' \cdot l_{sub} + l_{sub} + id_i^{min}.z' \cdot l) \\
v'_{j_{min}} &= (id_j^{min}.x' \cdot l, id_j^{min}.y' \cdot l, id_j^{min}.z' \cdot l) \\
v'_{j_{max}} &= (sid_j^{max}.x' \cdot l_{sub} + l_{sub} + id_j^{min}.x' \cdot l, \\
&\quad sid_j^{max}.y' \cdot l_{sub} + l_{sub} + id_j^{min}.y' \cdot l, \\
&\quad sid_j^{max}.z' \cdot l_{sub} + l_{sub} + id_j^{min}.z' \cdot l) \tag{Eq. A.3}
\end{aligned}$$

The x' , y' , z' index values of id^{min} can be obtained by Reverse function, while those for sid^{max} can be obtained by applying Reverse function to the sub-cells, i.e., using $N_{x_{sub}}$ and $N_{y_{sub}}$. Then, the extreme distances are computed from $(v'_{i_{min}}, v'_{i_{max}})$ and $(v'_{j_{min}}, v'_{j_{max}})$ using Equations Eq. 4.10-Eq. 4.13.

Appendix B

Parameter Settings for Grid Based Location Summarization

We provide the parameter combinations for $\frac{|D'|}{|D|} = \frac{1}{120}$ and $\frac{|D'|}{|D|} = \frac{1}{12}$, as shown in Tables B.1 and B.2 respectively. The parameter combinations in boldface are the ones producing the smallest T_2 . For example, when $\frac{|D'|}{|D|} = \frac{1}{120}$, GSLS yields the smallest T_2 when $|id| = 4$ bytes, $|\vec{r}| = 2$ bytes, and $w = 60$.

Table B.1: Parameter Settings for $\frac{|D'|}{|D|} = \frac{1}{120}$ (0.83%)

SLS	$w = 160$					
CLS	$w = 240$					
GCLS	$ id (\text{Bytes})$		w		l	
	1		20		84	
	2		40		13	
	3		60		2	
	4		80		0.33	
GSLS	$ id (\text{Bytes})$	$ \overline{r} (\text{Bytes})$	w	d_w	γ	l
	1	1	20	1806	3.53	84
	2	1	30		3.53	13
		2	40		0.014	
	3	1	40		3.53	2
		2	50		0.014	
		3	60		0.000054	
	4	1	50		3.53	0.33
		2	60		0.014	
		3	70		0.000054	
		4	80		0.0000002	
GSLS2	$ id (\text{Bytes})$	$ \overline{r} (\text{Bytes})$	w	d_w	γ	l
	1	1	20	203	0.396	84
	2	1	30	211	0.412	13
		2	40	217	0.0017	
	3	1	40	217	0.423	2
		2	50	219	0.0017	
		3	60	217	0.000007	
	4	1	50	219	0.428	0.33
		2	60	217	0.0017	
		3	70	216	0.000007	
		4	80	220	0.00000003	
GCLS2	$ id (\text{Bytes})$	$ sid (\text{Bytes})$	w	d_w	l_{sub}	l
	1	1	20	203	25.25	84
	2	1	30	211	25.91	13
		2	40	217	4.16	
	3	1	40	217	26.4	2
		2	50	219	4.2	
		3	60	217	0.655	
	4	1	50	219	26.56	0.33
		2	60	217	4.16	
		3	70	216	0.653	
		4	80	220	0.105	

Table B.2: Parameter Settings for $\frac{|D'|}{|D|} = \frac{1}{12}$ (8.33%)

SLS	$w = 16$					
CLS	$w = 24$					
GCLS	$ id (\text{Bytes})$		w		l	
	1		2		84	
	2		4		13	
	3		6		2	
	4		8		0.33	
GSLS	$ id (\text{Bytes})$	$ \overline{r} (\text{Bytes})$	w	d_w	γ	l
	1	1	2	1806	3.53	84
	2	1	3		3.53	13
		2	4		0.014	
	3	1	4		3.53	2
		2	5		0.014	
		3	6		0.000054	
	4	1	5		3.53	0.33
		2	6		0.014	
		3	7		0.000054	
		4	8		0.0000002	
GSLS2	$ id (\text{Bytes})$	$ \overline{r} (\text{Bytes})$	w	d_w	γ	l
	1	1	2	40	0.079	84
	2	1	3	56	0.110	13
		2	4	78	0.0006	
	3	1	4	78	0.153	2
		2	5	98	0.0008	
		3	6	117	0.000004	
	4	1	5	98	0.192	0.33
		2	6	117	0.0009	
		3	7	124	0.000004	
		4	8	135	0.00000002	
GCLS2	$ id (\text{Bytes})$	$ sid (\text{Bytes})$	w	d_w	l_{sub}	l
	1	1	2	40	6.3	84
	2	1	3	56	8.82	13
		2	4	78	1.94	
	3	1	4	78	12.29	2
		2	5	98	2.44	
		3	6	117	0.434	
	4	1	5	98	15.44	0.33
		2	6	117	2.76	
		3	7	124	0.451	
		4	8	135	0.076	

Appendix C

List of Publications

The work in this thesis contributed a total of 3 international publications with another awaiting review results as this thesis is prepared. The list is given below.

Conference Publications

- Ee-Peng Lim, Yida Wang, Kok-Leong Ong, and San-Yih Hwang. In Search of Knowledge About Mobile Users. In *ERCIM News No. 54 – Applications and Service Platforms for the Mobile User*, Jul. 2003.
- Yida Wang, Ee-Peng Lim and San-Yih Hwang. On Mining Group Patterns of Mobile Users. *Proc. of the 14th International Conference on Database and Expert Systems Applications - DEXA 2003*, Prague, Czech Republic, Sep. 2003.
- Yida Wang, Ee-Peng Lim and San-Yih Hwang. Efficient Group Pattern Mining Using Data Summarization. In *Proc. of the 9th International Conference on Database Systems for Advanced Applications - DASFAA 2004*, Jeju Island, KOREA, Mar. 2004.

Journal Publications

- Yida Wang, Ee-Peng Lim and San-Yih Hwang. Efficient Mining of Group Patterns from User Movement Data. *Data and Knowledge Engineering*, 2005.

References

- [AA03] L. A. Adamic and E. Adar. Friends and Neighbors on the Web. *Social Networks*, 25(3):211–230, 2003.
- [AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. In *Proc. 4th Int. Conf. Foundations of Data Organization and Algorithms*, Chicago, 1993.
- [AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules Between Sets of Items in Large Databases. In *Proc. of the ACM Int. Conf. on Management of Data*, Washington, USA, May 1993.
- [ALSS95] R. Agrawal, K.I. Lin, H.S. Sawhney, and K. Shim. Fast Similarity Search in The Presence of Noise, scaling, and Translation in Time-Series Databases. In *Proc. of VLDB'95*, Zurich, Switzerland, 1995.
- [AM70] J.G. Auguston and J. Minker. An Analysis of Some Graph Theoretical Cluster Techniques. *J. ACM*, Vol. 17:571–588, 1970.
- [AS94] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc. of the 20th Int. Conf. on Very Large Databases*, pages 487–499, Santiago, Chile, aug 1994.
- [AS95] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. 11th Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, 1995.
- [Bay98] J.R.J. Bayardo. Efficiently Mining Long Patterns from Databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 85–93. ACM Press, 1998.

REFERENCES

- [BCG01] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A Maximal Frequent Item-set Algorithm for Transactional Databases. In *Proc. of the 17th International Conference on Data Engineering*, Heidelberg, Germany, April 2001.
- [BD02] A. Bhattacharya and S. K. Das. LeZi-Update: An Information-Theoretic Framework for Personal Mobility Tracking in PCS Networks. *ACM/Kluwer Journal on Wireless Networks*, 8(2-3), Mar-May 2002.
- [BK73] C. Bron and J. Kerbosch. Algorithm 457: Finding All Cliques of an Undirected Graph. *Commun. ACM*, Vol. 16:575–577, 1973.
- [Boi74] J. Boissevain. *Friends of Friends: Networks, Manipulators, and Coalitions*. Oxford, 1974.
- [Cas95] W.F. Cascio. Whither Industrial and Organizational Psychology in a Changing World of Work. *American Psychologist*, 50:928–939, 1995.
- [CH00] D.J. Cook and L.B. Holder. Graph-Based Data Mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.
- [CK00] G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Dartmouth Computer Science Technical Report TR2000-381, Department of Computer Science, Dartmouth College, 2000.
- [CY05] X. Chai and Q. Yang. Reducing the Calibration Effort for Location Estimation Using Unlabeled Samples. In *3rd Annual IEEE International Conference on Pervasive Computing and Communications (IEEE Percom 2005)*, Mar 2005.
- [DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
- [DR01] G.M. Djuknic and R.E. Richton. Geolocation and Assisted GPS. *IEEE Computer*, 34(2):123–125, 2001.
- [EFKS98] M. Ester, A. Frommelt, H.P. Kriegel, and J. Sander. Algorithms for Characterization and Trend Detection in Spatial Databases. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining*, pages 44–50, New York City, 1998.

REFERENCES

- [EFKS00] M. Ester, A. Frommelt, H. P. Kriegel, and J. Sander. Spatial Data Mining: Database Primitives, Algorithms and Efficient DBMS Support. *Data Mining and Knowledge Discovery*, 4(2):193–216, July 2000.
- [EKS97] M. Ester, H.P. Kriegel, and J. Sander. Spatial Data Mining: A Database Approach. In *Proc. of Fifth Int. Symp. on Large Spatial Databases*, Berlin, Germany, 1997.
- [EK SX96] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996.
- [EKX95] M. Ester, H.P. Kriegel, and X.W. Xu. Knowledge Discovery in Large Spatial Databases - Focusing Techniques for Efficient Class Identification. In *Proc. 4th Int. Symp. on Advances in Spatial Databases*, pages 67–82, Portland, Maine, USA, 1995.
- [Feh04] K. Fehrenbacher. Secufone: The GPS Security Phone for Grandparents. <http://gps.engadget.com/entry/5718473382759249>, 2004.
- [FGNS00] L. Forlizzi, R.H. Guting, E. Nardelli, and M. Schneider. A Data Model and Data Structures for Moving Objects Databases. *ACM SIGMOD Record*, 29(2):319–330, May 2000.
- [Fis93] S.T. Fiske. Controlling Other People: The Impact of Power on Stereotyping. *American Psychologist*, 48:621–628, 1993.
- [FMT04] C. Faloutsos, K. McCurley, and A. Tomkins. Fast Discovery of Connection Subgraphs. In *The Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2004.
- [For96] D.R. Forsyth. Why So Social An Animal: The Functions of Groups. Colloquium presented at Virginia Commonwealth University, March 1996.
- [For99] D.R. Forsyth. *Group Dynamics*. Wadsworth, Belmont, CA, 3 edition, 1999.

REFERENCES

- [Fre92] L.C. Freeman. The Sociological Concept of “Group”: An Empirical Test of Two Models. *The American Journal of Sociology*, 98(1):152–166, July 1992.
- [GHW97] L. Garton, C. Haythornthwaite, and B. Wellman. Studying Online Social Networks. *Journal of Computer Mediated Communication*, 3(1), 1997.
- [GKT02] G. Giaglis, P. Kourouthanasis, and A. Tsamakos. *Mobile Commerce: Technology, Theory, and Applications*, chapter Towards a Classification Network for Mobile Location Services. MENNECKE, B.E. AND STRADER, T.J. (EDS.), Idea Group Publishing, 2002.
- [HB01] J. Hightower and G. Borriello. A Survey and Taxonomy of Location Systems for Ubiquitous Computing. Technical Report UW-CSE 01-08-03, University of Washington, Computer Science and Engineering, Box 352350, Seattle, WA 98195, August 2001.
- [HCH94] R. Hogan, G.J. Curphy, and J. Hogan. What We Know about Leadership: Effectiveness and Personality. *American Psychologist*, 49:493–504, 1994.
- [HDY99] J. Han, G. Dong, and Y. Yin. Efficient Mining of Partial Periodic Patterns in Time Series Database. In *Proc. of 15th Int. Conf. on Data Engineering*, pages 106–115, 1999.
- [HG02] X. Hong and M. Gerla. Dynamic Group Discovery and Routing in Ad Hoc Networks. In *Proceedings of the First Annual Mediterranean Ad Hoc Networking Workshop (Med-hoc-Net 2002)*, Sardegna, Italy, September 2002.
- [HGY98] J. Han, W. Gong, and Y. Yin. Mining Segment-Wise Periodic Patterns in Time-Related Databases. In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining*, pages 214–218, 1998.
- [HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.
- [HK02] Y. Huh and C. Kim. Group-based Location Management Scheme in Personal Communications Networks. In *16th International Conference on Information Networking (ICOIN-16)*, Korea, Jan 2002.

REFERENCES

- [Hoo04] A.V. Hooijdonk. GPS Personal Location Tracking Devices. <http://www.gps-practice-and-fun.com/personal-location-tracking.html>, 2004.
- [HP96] J.L. Han and A.W. Plank. Background for Association Rules and Cost Estimate of Selected Mining Algorithms. In *Proceedings of the Fifth International Conference on Information and Knowledge Management*, pages 73–80. ACM Press, 1996.
- [HP00] J. Han and J. Pei. Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. *ACM SIGKDD Explorations (Special Issue on Scalable Data Mining Algorithms)*, 2(2), December 2000.
- [HPY00] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns Without Candidate Generation. In *Proc. of Int. Conf. on Management of Data*, Dallas, TX, may 2000.
- [HPYM04] J. Han, J. Pei, Y. Yin, and R. Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.
- [HR57] F. Harary and I.C. Ross. A Procedure for Clique Detection Using the Group Matrix. *Sociometry*, Vol. 20:205–215, 1957.
- [Joh76] H.C. Johnston. Cliques of a Graph: Variations on the Bron-Kerbosch Algorithm. *Int. J. Comput. Inform. Sci.*, Vol. 5:209–238, 1976.
- [JYN05] Q. Yang J. Yin and L. Ni. Adaptive Temporal Radio Maps for Indoor Location Estimation. In *3rd Annual IEEE International Conference on Pervasive Computing and Communications (IEEE Percom 2005)*, Mar 2005.
- [KH95] K. Koperski and J. Han. Discovery of Spatial Association Rules in Geographic Information Databases. In *Proc. of 4th Int. Symp. on Advances in Spatial Databases*, pages 47–66, Portland, Maine, USA, 1995.
- [KKT03] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing The Spread of Influence Through a Social Network. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM Press, 2003.

REFERENCES

- [KMJ01] J.H. Kaufman, J. Myllymaki, and J. Jackson. IBM Almaden Research Center. <http://www.alphaworks.ibm.com/tech/citysimulator>, December 2001.
- [KZ01] Karam and M.J. Zaki. Efficiently Mining Maximal Frequent Itemsets. In *Proc. of 2001 IEEE International Conference on Data Mining (ICDM'01)*, San Jose, California, November 2001.
- [LC03] S.D. Lin and H. Chalupsky. Unsupervised Link Discovery in Multi-relational Data via Rarity Analysis. In *the Third IEEE International Conference on Data Mining (ICDM'03)*, 2003.
- [LC04] S.D. Lin and H. Chalupsky. Issues of Verification for Unsupervised Discovery Systems. In *KDD04 Workshop on Link Discovery*, Aug 2004.
- [Lev94] H. Levinson. Why the Behemoths Fell: Psychological Roots of Corporate Failure. *American Psychologist*, 49:428–436, 1994.
- [LLC04] G.H.K. Lam, H.V. Leong, and S.C.F. Chan. GBL: Group-based Location Updating in Mobile Wireless Environment. In *9th International Conference on Database Systems for Advanced Applications (DASFAA 2004)*, Korea, March 2004.
- [LT81] E. Loukakis and C. Tsouros. A Depth First Search Algorithm to Generate the Family of Maximal Independent Sets of a Graph Lexicographically. *Computing*, Vol. 27:249–266, 1981.
- [MC72] G.D. Mulligan and D.G. Corneil. Corrections to Bierstone’s Algorithm for Generating Cliques. *J. ACM*, Vol. 19:244–247, 1972.
- [MGJ02] S. Mathur, P. Gupta, and N. Jain. GPS Technology and Its Applications. In *The Asian GPS conference*, India International Centre, New Delhi, India, 24 - 25 October 2002.
- [MH01] S. Ma and J.L. Hellerstein. Mining Partially Periodic Event Patterns with Unknown Periods. In *Proc. 17th Int. Conf. on Data Engineering*, pages 205–214, Heidelberg, Germany, 2001.

REFERENCES

- [MH04] M. Mukherjee and L.B. Holder. Graph-based Data Mining on Social Networks. In *Workshop on Link Analysis and Group Detection (LinkKDD2004)*, Aug 2004.
- [NH94] R.T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proc. 20th Int. Conf. on Very Large Data Bases*, pages 144–155, Santiago, Chile, 1994.
- [ORS98] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic Association Rules. In *Proc. of the 14th Int. Conf. on Data Engineering*, pages 412–421, Orlando, Florida, USA, Feb 1998.
- [PBTL99] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules. In *Proc. of the 7th International Conference on Database Theory*, pages 398–416. Springer-Verlag, 1999.
- [PCY95] J. S. Park, M.S. Chen, and P.S. Yu. An Effective Hash-Based Algorithm for Mining Association Rules. In *Proc. of the ACM Int. Conf. on Management of Data*, San Jose, CA, USA, 1995.
- [PCY97] J. S. Park, M.S. Chen, and P. S. Yu. Using a Hashed-Based Method with Transaction Trimming and Database Scan Reduction for Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):813–825, October 1997.
- [PHA⁺01] J. Pei, J. Han, B.M. Asl, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In *Proceedings of 17th International Conference on Data Engineering (ICDE’01)*, pages 215–226, 2001.
- [PHA⁺04] J. Pei, J. Han, B.M. Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.C. Hsu. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(10), 2004.
- [PHM00] J. Pei, J. Han, and R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In *Proc. of 2000 ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery (DMKD’00)*, Dallas, TX, May 2000.

REFERENCES

- [PHW02] J. Pei, J. Han, and W. Wang. Mining Sequential Patterns with Constraints in Large Databases. In *Proceedings of 2002 Int. Conf. on Information and Knowledge Management (CIKM'02)*, 2002.
- [PSZ99] C. Parent, S. Spaccapietra, and E. Zimnyi. Spatio-temporal Conceptual Models: Data Structures + Space + Time. In *Proc. of 7th Int. Symp. on Advances in geographic information systems*, pages 26–33, Kansas City, MO USA, 1999.
- [Res02] Reed Electronics Research. RER- The Mobile Phone Industry - A Strategic Overview, October 2002.
- [RHH01] G.C. Roman, Q. Huang, and A. Hazemi. Consistent Group Membership in Ad Hoc Networks. In *International Conference on Software Engineering*, pages 381–388, 2001.
- [RHS00] J.F. Roddick, K. Hornsby, and M. Spiliopoulou. An Updated Bibliography of Temporal, Spatial and Spatio-Temporal Data Mining Research. In *Proc. of 1st Int. Workshop on Temporal, Spatial and Spatio-Temporal Data Mining*, Lyon, France, 2000.
- [RL01] J.F. Roddick and B.G. Lees. Paradigms for Spatial and Spatio-Temporal Data Mining. *Geographic Data Mining and Knowledge Discovery*, 2001.
- [Rym92] R. Rymon. Search through Systematic Set Enumeration. In *Proc. of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992.
- [SA96] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, volume 1057, pages 3–17. Springer-Verlag, 25–29 1996.
- [Sel99] T. Sellis. Research Issues in Spatio-Temporal Database Systems. In *Proc. of 6th Int. Symp. on Advances in Spatial Databases*, pages 5–11, Hong Kong, China, 1999.
- [SF99] M.D. Stephen and B. Frank. *Mobile Positioning*. Mobile Lifestreams, 1999.

REFERENCES

- [SH01] S. Shekhar and Y. Huang. Discovering Spatial Co-location Patterns: A Summary of Results. *Lecture Notes in Computer Science*, 2121:236–256, 2001.
- [She04] K. Sheikh. Predicting Customer Behaviour. *Express Computer*, April 2004.
- [SKKL98] E.J. Son, I.S. Kang, T.W. Kim, and K.J. Li. A Spatial Data Mining Method by Clustering Analysis. In *Proc. 6th Int. Symp. on Advances in geographic information systems*, pages 157–158, Washington D. C., USA, 1998.
- [SKR01] J.B. Schafer, J.A. Konstan, and J. Riedl. E-Commerce Recommendation Applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.
- [SON95] A. Savasere, E. Omiecinski, and S. B. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proc. of the 21th Int. Conf. on Very Large Databases*, pages 432–444, 1995.
- [TIAS77] S. Tsukiyama, M. Ide, H. Aviyoshi, and I. Shirakawa. A New Algorithm for Generating All the Maximum Independent Sets. *SIAM J. Comput.*, Vol. 6:505–517, 1977.
- [Toi96] H. Toivonen. Sampling Large Database for Association Rules. In *Proc. of the 22nd Conf. on Very Large Databases*, pages 134–145, Mumbai, India, September 1996.
- [TTT88] E. Tomita, A. Tanaka, and H. Takahashi. The Worst-case Time Complexity for Finding All the Cliques. *Technical Report UEC-TR-C5*, 1988.
- [Up03] Upoc.com. <http://www.upoc.com/corp/news/news-emarketer.html>, February 2003.
- [VKG02] M. Vlachos, G. Kollios, and D. Gunopoulos. Discovering Similar Multidimensional Trajectories . In *Proceedings of 18th International Conference on Data Engineering (ICDE’02)*, San Jose, California, 2002.
- [VVK00] U. Varshney, R. Vetter, and R. Kalakota. Mobile commerce: A New Frontier. *IEEE Computer: Special Issue on E-commerce*, pages 32–38, October 2000.
- [WF94] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press, 1994.

REFERENCES

- [WHP03] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets. In *Proc. of 2003 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'03)*, Washington, D.C., Aug 2003.
- [WLH03] Yida Wang, Ee-Peng Lim, and San-Yih Hwang. On Mining Group Patterns of Mobile Users. In *Proc. of the 14th International Conference on Database and Expert Systems Applications - DEXA 2003*, Prague, Czech Republic, 1-5 Sep 2003.
- [WLH04] Yida Wang, Ee-Peng Lim, and San-Yih Hwang. Efficient Group Pattern Mining Using Data Summarization. In *Proc. of the 9th International Conference on Database Systems for Advanced Applications - DASFAA 2004*, Jeju Island, KOREA, March 17-19 2004.
- [WYY00] W. Wang, J. Yang, and P.S. Yu. Mining Patterns in Long Sequential Data with Noise. *SIGKDD Explorations*, 2(2):28–33, 2000.
- [WYY01a] W. Wang, J. Yang, and P.S. Yu. Infominer: Mining Surprising Periodic Patterns. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 395–400, San Francisco, California, August 2001.
- [WYY01b] W. Wang, J. Yang, and P.S. Yu. Meta-patterns: Revealing Hidden Periodic Patterns. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, pages 550 –557, 2001.
- [WYY02] W. Wang, J. Yang, and P.S. Yu. InfoMiner+: Mining Partial Periodic Patterns with Gap Penalties. In *Proc. of the 2nd IEEE International Conference on Data Mining (ICDM)*, 2002.
- [YP03] Y. Yang and B. Padmanabhan. Segmenting Customer Transactions Using a Pattern-Based Clustering Approach. *Third IEEE International Conference on Data Mining*, pages 19 – 22, November 2003.

REFERENCES

- [YWW00] J. Yang, W. Wang, and P.S. Yu. Mining Asynchronous Periodic Patterns in Time Series Data. In *Knowledge Discovery and Data Mining*, pages 275–279, 2000.
- [YWW02] J. Yang, W. Wang, and P.S. Yu. Mining Asynchronous Periodic Patterns in Time Series Data. *IEEE Transaction on Knowledge and Data Engineering (TKDE)*, 2002.
- [YYWH02] J. Yang, P. Yu, W. Wang, and J. Han. Mining Long Sequential Patterns in a Noisy Environment. In *Proceedings of 2002 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD’02)*, 2002.
- [Zak00a] M.J. Zaki. Generating Non-Redundant Association Rules. In *Proc. of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, MA, August 2000.
- [Zak00b] M.J. Zaki. Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May/June 2000.
- [Zak01] M.J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning Journal, special issue on Unsupervised Learning (Doug Fisher, ed.)*, 42:31–60, 2001.
- [Zar96] P. Zarchan. *Global Positioning System: Theory and Applications*, volume I. American Institute of Aeronautics and Astronautics, 1996.
- [ZG01] M. J. Zaki and K. Gouda. Fast Vertical Mining Using Diffsets. *RPI Technical Report*, 1(1), 2001.
- [ZGL03] V. Zeimpekis, G.M. Giaglis, and G. Lekakos. A Taxonomy of Indoor and Outdoor Positioning Techniques for Mobile Location Services. *SIGecom Exch.*, 3(4):19–27, 2003.
- [ZH02] M.J. Zaki and C.J. Hsiao. CHARM: An Efficient Algorithm for Closed Item-set Mining. In *Proc. of 2nd SIAM International Conference on Data Mining*, Arlington, April 2002.
- [Zha02] Y. Zhao. Standardization of Mobile Phone Positioning for 3G Systems. *IEEE Communications*, 40(7), July 2002.