Algorithmic Approaches to Pattern Mining from Structured Data

Keisuke Otaki
Doctoral dissertation

Department of Intelligence Science and Technology
Graduate School of Informatics
Kyoto University

A doctoral dissertation

submitted for the degree of Doctor of Informatics.

Department of Intelligence Science and Technology,
Graduate School of Informatics,
Kyoto University.

Typeset with
XeLaTeX3.14159265-2.6-0.99992 (TeX Live 2015)

Abstract

Data Mining is conception related to methods for discovering previously unknown knowledge in data to realize Knowledge Discovery. The flow to achieve the discovery is named KDD process, and often modeled as consecutive problems: pre-processing of data, discovering knowledge from the refined data with some algorithms, and post-processing of the discovery. Pattern mining is a principal research topic to implement the step of discovering knowledge, where researchers aim at developing methodologies and theories of finding combinatorial patterns satisfying given constraints. Many researchers have shown an interest in both exploring variations of patterns and developing algorithms for pattern mining in the last 25 years. In the beginning of Data Mining, numerical / categorical data in table format were typical objects. However, rapid growing technologies concerning the Internet and computer science generate a large quantity of graphs in a scientific discipline, and Knowledge Discovery from graphs have become much important in these days. Therefore, nowadays the field is also focusing on a bunch of combinatorial objects (e.g., sequences, trees, and graphs).

This thesis investigates pattern mining and its surrounding problems with methods and theories for algorithms to enhance the usability and the effectiveness of pattern mining. Main objects for the KDD process treated in the thesis are graphs. Nowadays previously developed pattern mining algorithms work efficiently due to sophisticated enumeration algorithms. They are able to list all frequent patterns from large databases, regardless of whether they are theoretically guaranteed. However, in many problems, applying enumeration algorithms to large scale data is still challenging in the sense that they generate a huge amount of patterns which could be larger than the size of the input and redundant. This phenomenon is known as pattern combinatorial explosions. Methods to overcome the explosions still remains controversial. In order to utilize pattern mining with large scale and complex data, more technical advances are required, and we would contribute it with studying both theoretical and practical aspects of pattern mining. More precisely, we build novel methods *from pre-processing to post-processing* for graphs generated by various manners in pattern mining.

First we answer the fundamental question: What is the theoretical efficiency of mining algorithms? We discuss the listing complexity of mining algorithms for graphs, particularly for graphs of bounded treewidth – a parameterized graph class –, to clarify the performance and the theoretical limitations of algorithms. Based on developments of efficient algorithms including the contribution above, we investigate practical aspects of pattern mining, by focusing on both post-processing and pre-processing of pattern mining. Second, we prepare a method to support exploratory data analysis by users in post-processing of pattern mining. We build a kind of structure, lattices of enumerated patterns. With the lattice structures we develop a new formalization based on the minimum description length principle, to solve the unsupervised pattern set mining problem, which is known as a major problem for post-processing to reduce the size of the output. Third, we investigate pre-processing of pattern mining to enhance the readability of enumerated

patterns based on clustering. We evaluate the effect of the pre-processing method from the output of pattern mining. The developed methods above are experimentally evaluated and discussed from the viewpoint of how they are helpful in pattern mining via graphs. Our experimental results indicate that the developments of methods for pattern mining from pre-processing to post-processing are valuable to enhance the utility, the usability, and the effectiveness of pattern mining.

Keywords: frequent pattern mining, graphs of bounded treewidth, formal concept analysis, pattern structure, lattice, generalized pattern, unsupervised pattern set mining, minimum description length principle, edit operation, similarity graph, data pre-processing, pattern frequency spectrum.

Acknowledgments

I am deeply grateful to all the people who have supported me during my Ph.D course. First of all, I would like to thank my supervisor Prof. Akihiro Yamamoto. His enormous support and insightful comments were invaluable to my researches. I would also like to owe my deepest gratitude to the other committee members, Prof. Tatsuya Akutsu and Prof. Hisashi Kashima, for reviewing this thesis and for discussions to improve the thesis.

In addition, I would like to show my greatest appreciation to my co-authors. Particularly, Dr. Tamás Horváth and Dr. Jan Ramon gave me enormous advice, which leads me to the main research field in the thesis, i.e., pattern mining from structured data. I would also like to thank Dr. Mahito Sugiyama, Dr. Ryo Yoshinaka, Mr. Madori Ikeda, Mr. Kentaro Imajo, Mr. Seiichi Kondo, and Mr. Tomohiko Okayama, who were also my co-authors related to Yamamoto Cuturi Laboratory. Discussions with them were valuable to me when completing the thesis and promoting researches in my Ph.D course.

I would also like to express the deepest appreciation to the member of Yamamoto Cuturi Laboratory for their useful comments regarding my research. Both professors Dr. Cuturi Marco and Dr. Ryo Yoshinaka were sincerely supporting my Ph.D course in the laboratory. With Dr. Cuturi, I had a good experience to organize Machine Learning Summer School held in Kyoto twice in 2012 and in 2015, which was one of the most fascinating events during the course. My deepest appreciation also goes to members in the CAML group of Fraunhofer IAIS. I received generous support from them during my stay in Germany.

Apart from the researches, I gratefully appreciate the financial support of the KDDI foundation (for my stay in Germany), the JSPS (Japan Society for the Promotion of Science), and the JASSO (Japan Student Services Organization) that made it possible to complete my thesis.

Finally, I would like to express my gratitude to my parents and my brother for their moral support and warm encouragements to complete my Ph.D course.

Contents

Listing of Figures

List of Tables

All truths are easy to understand once they are discovered; the point is to discover them.

Galileo Galilei (1564--1642)

CHAPTER 1

Introduction

DATA MINING is a principal research field concerned with discovering *knowledge* from databases. With increasing the size and the variety of data, the field has attracted much attention in the last 25 years. In the field researchers aim at developing methodologies to find *unknown* but *potential useful* knowledge in the form of patterns, rules, subgroups, etc. from databases and theories to support the discovery. Although the goal – to find *something useful from data* – is a bit abstract, great success of many advances and developed applications have been widely recognized, and now *data science* has become one of the most fascinating scientific disciplines in both research societies and industries. A typical example of definitions of Data Mining collected by Friedman[Fri97] is that Data Mining is *the process of extracting previously unknown, comprehensive, and actionable information from large databases and using it to make crucial business decisions* (by Zekulin). Another example is that Data Mining is *a set of methods used in the knowledge discovery process to distinguish previously unknown relationships and patterns within data* (by Ferruzza).

Data Mining

Data Science

Recent success of studies in Data Mining are supported by rapid growths of related technologies, including high performance computation, networks and distributed systems, databases, data management, artificial intelligence, and machine learning. Those areas also generate various data, characterized by the word 3V (*volume*, *variety*, and *velocity*) of data, which is now recognized as fundamental conception of Big Data. Along with the increase of 3V of data, the research field of Data Mining has become valuable to find useful knowledge, and consequently the filed has attracted much attention. On the one hand, Data Mining from various data is undoubtedly important to users, and on the another hand it arises many challenges in both research societies and industries.

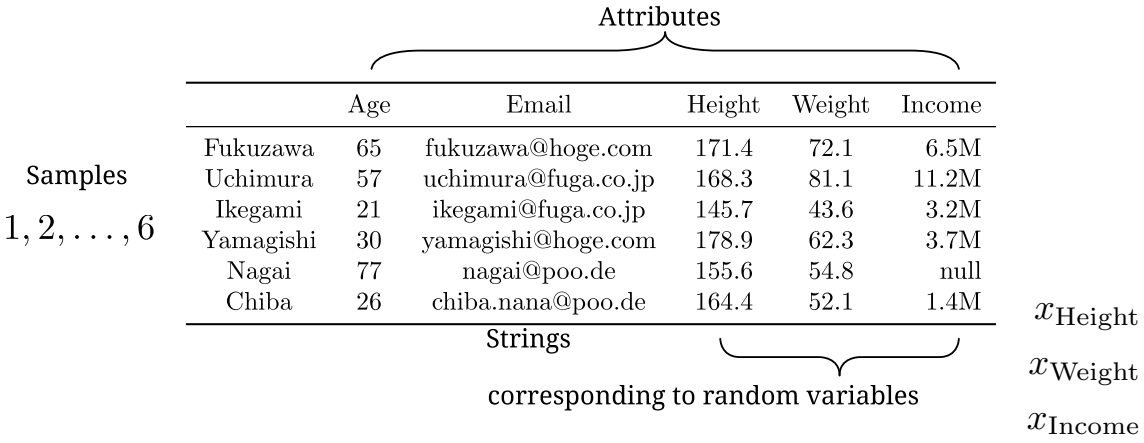
3V

Big Data

Originally in the traditional style in statistics, users focused on data represented by tables (a.k.a. relational data), including numerical or categorical vectors, as illustrated in Figure 1.1. Computed several statistics (e.g., mean, standard deviation, etc.) are discovered insights, which characterize the given table data. Furthermore, considering a (linear) model between three random variables x_{Height}, x_{Weight}, and x_{Income} and learning parameters of the model from samples are a typical example of statistical analyses.

Relational Data

In these days, however, data that are not represented as a table appear because of the increase of various applications concerned with the Internet. Examples of



	Age	Email	Height	Weight	Income
Fukuzawa	65	fukuzawa@hoge.com	171.4	72.1	6.5M
Uchimura	57	uchimura@fuga.co.jp	168.3	81.1	11.2M
Ikegami	21	ikegami@fuga.co.jp	145.7	43.6	3.2M
Yamagishi	30	yamagishi@hoge.com	178.9	62.3	3.7M
Nagai	77	nagai@poo.de	155.6	54.8	null
Chiba	26	chiba.nana@poo.de	164.4	52.1	1.4M

Figure 1.1.: Data in the table format, which is a set of tuples each of which corresponds to a sample. Each value in a vector indicates some meaningful measured value of an attribute. A basic task is considering a model f from attributes to the target attribute; e.g., discovering a function f of x_{Height} and x_{Weight} to predict x_{Income}.

such non-table format data include semi-structured documents (e.g., HTML/XML documents), graph-based documents for Knowledge-Base Systems (e.g., RDF documents), Bioinformatics / medical data, and social networks. According to the emergence of such structured data, developing methods for Data Mining with taking into account the structure of data has become much important, and several studies have been devoted in the last 15 years. A large part of such data can be represented in *graphs*, which are main objects throughout the thesis. Data represented by graphs are ubiquitous in these days. Particularly, we categorize graphs dealt in the thesis into two groups:

Graph

1. graphs explicitly generated or adopted in a problem domain, and
2. graphs implicitly adopted in a problem domain.

To explain the expressiveness and the utility of graph-based representations, we here introduce two scenarios of Data Mining from structured data.

Example 1: Discovery on Graphs Graphs can be used to directly represent various objects based on *relations* among *entities*. An *entity* corresponds to an (individual) data unit. A *relation* explains the relationship between entities with or without directions. In Graph Theory, V denotes the set of entities, called *vertices*, and E denotes the set of relations, called *edges*, and the pair $G = (V, E)$ denotes a graph, where $E \subseteq V \times V$, i.e., relations are defined with pairs of vertices. Let us consider the *following* and *follower* relationship in twitter[*]. Users correspond to vertices, and the following / follower relationship can be represented as edges. That is, graphs are directly representing the phenomena we are interested in. Figure 1.2 illustrates the graph drawn from the twitter data[†] with several tasks of Data Mining.

[*]Twitter, http://twitter.com/

[†]From http://www.martingrandjean.ch/introduction-to-network-visualization-gephi/ (accessed at Sep. 15, 2015.)

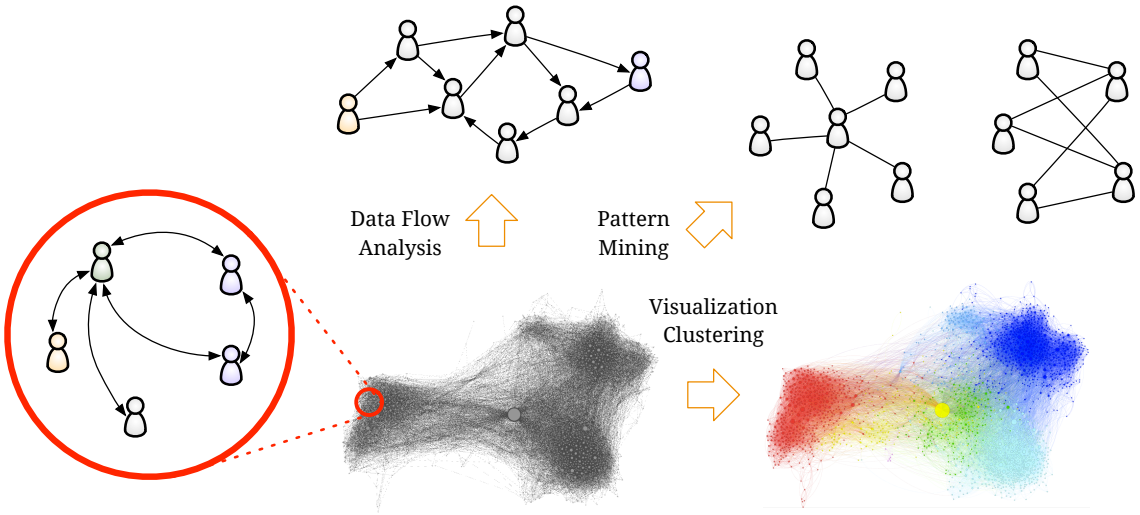


Figure 1.2.: A graph from twitter data. Vertices in V represent users, and edges in E represent the following / follower relationship. Then various phenomena observed can be modeled in Graph Theory; e.g., evaluating distances, finding characteristic graph structures, or detecting hidden communities and visualizations, etc.

Though "data analysis" is conception covering various problems, once given data are represented as graphs, tasks and methods to analyze the data can be formally described. The followings are examples.

- Link Mining (e.g., [CDK+99,GD05]); topics related to build predictive or descriptive models of edges on graphs.
- Graph Classifications (e.g., [KMM04,DKWK05]); supervised classification problems in which each datum is given as a graph.
- Social Network Analyses (e.g., [WF94,Sco12]); area to investigate network structures in the intersection area of graph theory, computer science, and sociology.
- Graph Pattern Mining (e.g., [IWM00,KK01,YH02]); methodologies of finding interesting substructures of graphs as several forms (e.g., paths, subtrees, subgraphs). For example, frequently appearing subgraphs show common structures behind the entire network or several networks.
- Community Detection (e.g., [New04,New06,For10]); finding *communities* among users to clarify structures of graphs. For example, modularity-based clustering of vertices, graph cuts, hierarchical clustering, finding cliques have been studied.
- Graph Kernels (e.g., [G03,GFW03,KTI03,VSKB10]); developing and investigating methods for comparing two graphs.
- Graph Indexing (e.g., [SWG02,YYH04]); methods for constructing data structures and indices for querying and answering some statistics on the graph. For example, answering the distance between two users, evaluating the property of communities in the graph, etc.
- Domain Specific Applications; dealing with biological data (e.g., [HYH+05,PTU10]), for example.

We have several possible formulations to realize the discovery from graphs according to problem and data. For example, a *community*; a potential meaningful

Table 1.1.: An example of transactions with timestamps representing an Web access log (generated by apache-loggen, and transformed for the thesis), consisting of four attributes Timestamp, User ID, IP Address, Visited Pages (a set of symbols representing pages).

Timestamp	User ID	IP Address	Visited Pages
22/Oct/2015:16:56:31	0012039	40.60.27.166	a, b, d
22/Oct/2015:16:56:48	0020061	192.201.225.220	a, e
22/Oct/2015:16:57:12	0010199	84.66.224.70	b, c, d
22/Oct/2015:16:58:01	0079180	164.87.21.44	c, f
22/Oct/2015:16:58:22	0191805	76.87.35.51	e, f, g

Clique
Connected Component

subgraph G' of the given graph G can be defined by several manners, including *cliques* (i.e., complete subgraphs of G), a set V' of vertices *inducing a dense subgraph of* G, *connected components* (i.e., subgraphs in which any two vertices are connected) of G. Several parameterization of graphs might be useful for discussions. Examples are including the width, the weight, the number of connected components, the average size of connected components, and the density of edges.

Timestamp

Example 2: Analyzing Ordered Transactions Using Graphs Let us see another example: transaction databases with timestamps. In databases, we often face *ordered* transactions by some attribute (e.g., times, unique IDs), where transactions can be sorted (e.g., in the chronological order). Table 1.1 shows examples, where we assume that databases have *timestamps* as its ordering auxiliary attribute of main information (e.g., a set of visited pages by a user in the right-most column). In such a situation, Data Mining taking into account the order have been attracted much attention to extract useful knowledge based on "time" induced by the order. A fundamental task is to analyze the *precedence-subsequence* relation of events when we have a set of events[‡]. The followings are examples studied in pattern mining from such databases with some ordering attribute.

Precedence-Subsequence Relation

Sequential Pattern

- Sequential patterns (e.g., SA96,Zak01,PHMa+01); finding linear combinations of the fixed set of symbols from various transaction databases.

Association Rule

- Association rules (a.k.a. Bipartite episodes, e.g., AS94,ZPOL97,KAH09); finding a pair (X, Y) of two sets of symbols from the set of symbols, which can be interpreted as a rule of the form $X \to Y$, where X and Y are a *premise* (e.g., conditions) and a *consequence* (e.g., results), respectively.

Episode Pattern

- Episodes (e.g., MTIV97,Kat11,TC12); discovering general rules from databases, formulated by *directed graphs* $G = (V, E)$, where labels of vertices in V can be modeled as a set of symbols and relations represented by edges in E are a bit generalization of the above two examples.

All of the above examples can be represented in graphs, and therefore mining these relations from databases *generate* many graphs *indirectly* in Data Mining. We can thus assume that methods for graphs represented directly (as in Example 1) can

[‡]The set of symbols, *alphabet*, is denoted by $\mathcal{B}$ in the thesis.

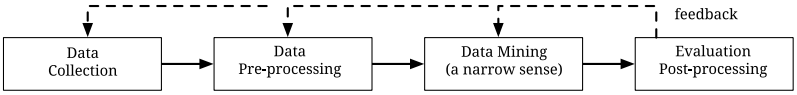


Figure 1.3.: The KDD process based on [FPsS96] consisting of the main four parts.

be applied to *indirectly generated* graphs. The key conception on the various problems in the thesis is that both *algorithms* and *mathematical formulations* that reflect properties of given data are important to clarify and achieve the purpose of Data Mining.

Throughout the two examples, we confirm that formulations using graphs are ubiquitous and widespread. It is not only for graphs explicitly given (Example 1) but also other data (e.g., transactions with timestamps) related to graphs indirectly (Example 2). From those graphs, the problem is how to discover knowledge.

Models of KDD An overall model of data analysis, i.e., *Knowledge Discovery in Databases* was described by Fayyad et al.[FPsS96] as below. KDD

> *Knowledge Discovery in Databases* (**KDD**) is the nontrivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in large databases[FPsS96].

An ideal idea of finding useful, hidden, and valuable knowledge from databases can be modeled as the KDD process, consisting of the four parts as illustrated in Figure 1.3:

1. **Data collection:** preparing databases in a structured, semi-structured, or non-structured fashion. In some application scenarios we need to integrate multiple databases or tables.

2. **Data pre-processing:** executing data pre-processing tasks listed below. Related topics are surveyed in a recent book[GLH15]. The tasks are essential to make the data suit for algorithms used in the later parts:
   - Data selection (e.g., removing redundant parts of the input).
   - Data discretization (e.g., methods based on uniform bins, histograms, etc.)
   - Data cleaning (e.g., removing noisy data, compensating missing data)
   - Data transformation (e.g., Linear / Quadratic / Polynomial Transformation of data, The Box-Cox method) for making the distribution of a datum suit for algorithms.

3. **Data mining:** doing a main task by applying a selected data analysis method (Supervised methods, unsupervised methods, parametric / non-parametric algorithms, parameter estimations, enumeration algorithms, etc.), to find *patterns* for the later decision making step, and

4. **Evaluation and post-processing**: evaluating, utilizing, and visualizing results of the main task for extracting knowledge from the selected data and to support users' decision making.

Note that in the thesis we adopt *pattern mining* in the main data mining part of the KDD process, assuming that labels are not give (i.e., unsupervised problem), to find useful but hidden knowledge in the form of (various) patterns. This assumption means that some refined data are given to pattern mining algorithms. More details and overview of pattern mining is given in Chapter 2.

For clear discussions on data analysis, i.e., to clarify the purpose and the problem of specific tasks from the viewpoint of the KDD process above, we often face the following questions: **(Input)** What are the data? **(Output)** What do users want to find from the input? **(Method)** How can algorithms compute the output? For example in Example 1, finding cliques is a way of accomplishing the *community detection* for Data Mining from graphs, summarized below:

> **Input** A labeled undirected graph $G = (V, E, \lambda)$
>
> **Output** All cliques in G whose size is larger than k
>
> **Method** Applying a clique enumeration algorithm to G, and prune the cliques listed if the size is less than or equal to k

From this viewpoint, the purpose of Data Mining is no longer *ambiguous*. It is clearly described and several methods can be applied to solve the problem. Then the questions become both theoretical and practical ones. Is the enumeration problem computationally tractable or intractable? Is the enumeration efficient? How much memory spaces are required for the enumeration? What kinds of graphs can be enumerated efficiently?

To give clear objectives, precise problem definitions, and develop efficient methods, both theoretical backgrounds and practical studies are important. For example to deal with graphs directly given (as in Example 1), *Graph Theory* supports us to handle them theoretically, and various practical and efficient algorithms developed in *Algorithmic Graph Theory* could support us to computationally solve specific problem instances on graphs for Data Mining. For analyzing ordered data (e.g., numbers, finite sets, or sequences, including transactions with timestamps, as in Example 2), *Order Theory* provides us mathematical formulations. Methods in the theory called *Formal Concept Analysis* is an algorithmic approach based on an order among data. Obviously both theoretical and practical studies are important for Data Mining from *data that have structures on them*.

Based on successes of various problems and developed algorithms for pattern mining, we should recognize that pattern mining often requires efficient enumeration algorithms. Therefore, it is often assumed that given data are well-formatted and cleand for enumeration algorithms, and consequently the input of pattern mining is also fixed (as illustrated in Figure 1.4). In addition, the algorithms are often applied once to databases, i.e., feedback is not taken into account, and their outputs are represented in mere lists of patterns. In such a situation, where enumeration

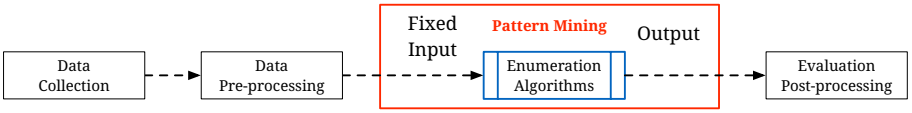
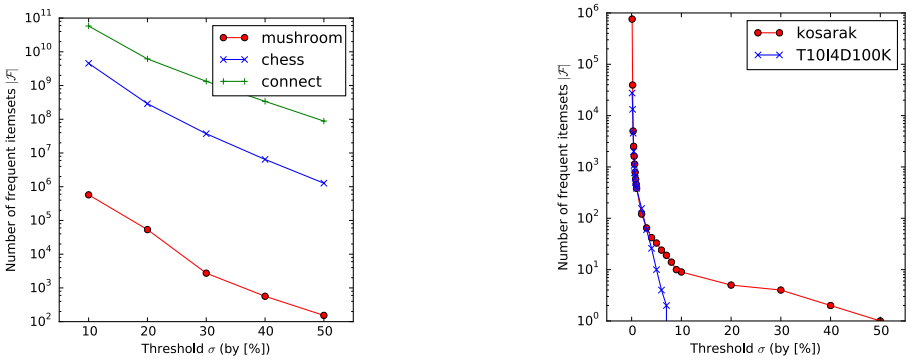


Figure 1.4.: Pattern mining in the KDD processing, where the input is fixed and feedback is usually not taken into account.



(a) From datasets mushroom, chess, and connect with varying $\sigma = 50\%, 40\%, \ldots, 10\%$.

(b) From datasets kosarak and T10I4D100K with varying $\sigma = 50\%, \ldots, 10\%, 9\%, \ldots, 1\%$, and $0.9\%, \ldots$.

Figure 1.5.: Numbers of frequent itemsets mined from several benchmark datasets with varying the threshold parameter, where the y-axis is the logarithmic scale.

algorithms are used to implement pattern mining from structured data, we commonly face the following problems.

- The size of the output of enumeration algorithms is often *larger* than the size of the input (a.k.a., pattern combinatorial explosions).
- The output is sensitive to hyperparameters.
- The output list is redundant and has no explict structures.

To see these problems we show small experimental results of the simplest pattern mining instance; *Frequent Itemset Mining*[AS94]. We adopted a freely available Itemset implementation[§] of the ECLAT algorithm (It will be explained more in Chapter 2), and used five benchmark datasets named mushroom, chess, connect, kosarak, and T10I4D100K. After applying mining algorithms to the datasets with varying the threshold parameter, we observed and plotted the number of frequent itemsets of them (i.e., the number of elements in the output), as illustrated in Figure 1.5, where Figure1.5a shows results of mushroom, chess, and connect, and Figure 1.5b does for kosarak and T10I4D100K. As the y-axes of them are in the logarithmic scale, we can easily confirm that the size of the output of pattern mining can possibly be exponential to the size of input, meaning that the size of the output could be larger than the size of the input. In addition, judging from results in Figure 1.5b, it is obvious that tuning hyperparameters (i.e., a threshold θ). Because these outputs are generated by enumeration algorithms, they are given as mere lists or sets of frequent itemsets in experiments. That is, they are redundant (i.e., some itemsets in

[§]See http://www.borgelt.net/fpm.html for experiments.

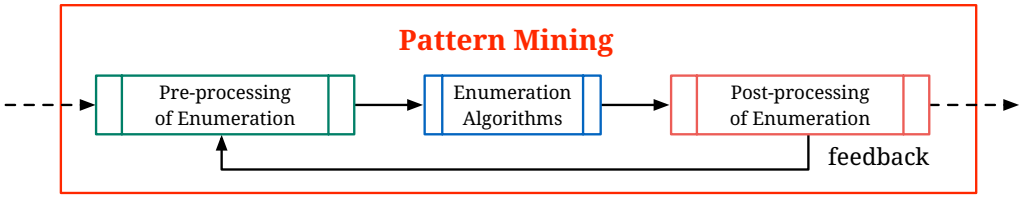


Figure 1.6.: Pattern mining with pre-processing, post-processing, and feedback.

the output are related to others each other) and have no explict structures. From these results, we recognize that methods to overcome the above problems should be important.

1.1. Contributions

This thesis would like to contribute for pattern mining in the KDD process by focusing on the above three problems. The key idea is focusing not only the core part of pattern mining with enumeration algorithms but also pre-processing and post-processing of pattern mining. In addition, by considering feedback from the output to its pre-processing, we can build an *iterative approach* (a.k.a. adaptive, exploratory, or interactive methods) as illustrated in Figure 1.6. On the conceptual sketch of the KDD process, we list contributions of the thesis, which consist of four main chapters. See also pp. 81 – 83 for the list of publications. Before the main contents, preliminaries are given in Chapter 2. This chapter covers important aspects of the problem; foundations, theory, and algorithms (Sections 2.1, 2.2, and 2.3). It also contains the list of related publications (Section 2.4).

Theoretical Aspects of Pattern Mining The results of this part have been published in [P4]. We first mainly focus on theoretical aspects on the *listing problem*, which is known as a general description of several pattern mining problems. On the listing problem, *listing complexities* are mainly discussed, where intuitively we discuss *how difficult listing processes are* for a given problem setting of pattern mining. Without any restrictions, pattern mining from graphs is relatively intractable and investigating the listing complexity is difficult.

Listing Complexity

In Chapter 3, the key conception is to *utilize common properties of a given target database*. In the chapter, targets are *chemical compound graphs*, characterized by the *treewidth* parameter, where compounds can be modeled as labeled undirected graphs whose *treewidth* is *typically less than or equal to 3* (in many cases the treewidth parameter is 2). The parameter *treewidth* is also known to be useful from algorithmic viewpoints as many decision problems on graphs can be efficiently solved if the treewidth is fixed and bounded by k. However, *induced subgraph isomorphism* remains computationally hard even though the treewidth parameter is a small constant (Sections 3.2 and 3.3). Apart from the algorithmic intractability, with *tree decompositions* and *dynamic programming approaches*, we can design the efficient mining algorithms (Section 3.4). They are summarized as follows.

Treewidth

Input A set $\mathcal{D}$ of (undirected) graphs from a graph class $\mathcal{G}$ of *bounded treewidth graphs* and a threshold parameter θ

Output A list of all frequent connected subgraphs on the class $\mathcal{G}$ that are induced subgraph isomorphic to at least θ graphs in $\mathcal{D}$

Method A dynamic programming-based levelwise search algorithm

Practical Aspects of Pattern Mining We next focus on practical aspects of pattern mining from viewpoints of pre-processing and post-processing. Results about post-processing in the first half are published in [P2, P3] and those of pre-processing in the second half are done in [J1,P1]. We begin our discussions on post-processing assuming that pattern mining is efficiently possible. We then develop pre-processing of pattern mining to address problems when pattern mining is possible but hard to tune its parameter. Throughout chapters, we focus on how we can utilize the result $\mathcal{F}$ of pattern mining based on the redundancy on $\mathcal{F}$. That is, the output, i.e., the list or set of (interesting) patterns, is usually *redundant* by meanings that some patterns subsume others, they are similar to others, and many patterns have similar degree of interestingness for users. To overcome this issue, we often need to take care of the *selection* of the whole patterns to utilize pattern mining as a core method of the KDD process.

Redundancy

(This chapter is closed. Only a summarization is provided in this online version) In Chapter 4, we focus on the problem of constructing *lattice structures* from the output $\mathcal{F}$ of graphs to support users' exploratory data analyses in the evaluation / post-processing part. Our idea is that many patterns in $\mathcal{F}$ are mutually related and are in the sibling relation on an enumeration tree. Such two patterns are almost the same, where a few sub-structures of patterns are different and most of them are shared. We would like to merge similar patterns into clusters to clarify the structure behind the set $\mathcal{F}$ using pattern structures. These methods are also discussed via computational experiments and related works are also explained.

Lattice
Exploratory Data Analysis
Pattern Structure

Input A list $\mathcal{F} = \langle G_1, G_2, \ldots, G_N \rangle$ where $G_i \in \mathcal{G}$ ($1 \leq i \leq N$) on some class $\mathcal{G}$ of episodes

Output A lattice *structure* $\langle \mathfrak{L}, \preceq \rangle$ comprehensively and compactly represents $\mathcal{F}$ with the order $\preceq$

Method Adopting *Lattice Theory* and its algorithmic methodology named *pattern structures*, which are our new formulations and applications particularly for episodes

Pattern Structure

(This chapter is closed. Only a summarization is provided in this online version) In Chapter 5, we revisit the key relation between pattern mining and lattice structures with two important concepts in data mining; *closed itemsets* and *formal concept analysis*. Based on lattices, we build a new interpretation of pattern set mining problems to give an evaluation method (i.e., post-processing) to overcome pattern combinatorial explosions using lattices and an Information Theory-based

Closed Itemset
Formal Concept Analysis

criteria. More precisely, we are focusing on problems to reduce the size of the output of pattern mining (i.e., compute a set $M \subseteq \mathcal{F}$ such that $|M|$ is small enough).

Pattern Set Mining

This problem has been known and studied as unsupervised pattern set mining.

> **Input** A lattice $\langle \mathfrak{L}, \preceq \rangle$ (or, a list $\mathfrak{L}$ of concepts)
>
> **Output** A sublattice structure $\langle \mathfrak{L}', \preceq \rangle$ satisfying $|\mathfrak{L}'| \leq |\mathfrak{L}|$ (i.e., a *selected* list $\mathfrak{L}'$ of concepts)
>
> **Method** A greedy algorithm based on the state-of-the-art and common criteria named *Two-part Minimum Description Length principle* to solve the unsupervised pattern set mining problem

Two-part MDL Principle

In Chapter 6, we discuss the *data pre-processing part* of pattern mining based on the fact that the result $\mathcal{F}$ depends on not only the threshold parameter θ but also the input database $\mathcal{D}$ and symbols used in $\mathcal{D}$. We study pre-processing of such a database $\mathcal{D}$ using similarity graphs to take into account co-occurrences of symbols (Sections 6.2 and 6.3). The method in this chapter aims at reduce the number of symbols used in $\mathcal{D}$ to realize effective pattern mining from $\mathcal{D}$. We also discuss the difference between mining results with and without pre-processing via computational experiments (Sections 6.4 and 6.5).

> **Input** A database $\mathcal{D} = \{\mathcal{S}\}$ of a single sequence $\mathcal{S}$
>
> **Output** A set of (generalized) partially periodic patterns (PPPs)
>
> **Method** A pre-processing using *similarity graphs* built from $\mathcal{D}$ and clustering on graphs to reduce the number of symbols

That is our pre-processing constructs a function f to transform the data sequence $\mathcal{S}$ to $\mathcal{S}'$ so that the transformed sequence $\mathcal{S}'$ is much easier than the original one $\mathcal{S}$ for pattern mining.

CHAPTER 2

Preliminaries

Many pattern mining problems can be discussed in a unified manner by considering an order among patterns. To explain basic concepts used in the thesis, we establish preliminaries and common notations for the following chapters in Sections 2.1, 2.2, and 2.3 with major references in the literature in Section 2.4. A list of symbols, notations, and mathematical backgrounds are also summarized in Appendix A.

Pattern mining, from a viewpoint of Data Mining, is a collection of problems aiming at finding combinatorial patterns (or features) from databases. Then an algorithm for pattern mining can be regarded as a tool in Data Mining to realize the discovery in Knowledge Discovery. We have various data in real applications, and regularities behind data can be modeled by different patterns. A famous instance of models capturing a regularity behind a database is *Frequent Itemset Mining* by Agrawal, Imieliński, and Swami[AIS93]. This is revisited by Agrawal and Srikant[AS94]. Following the work, many variations of patterns have been investigated.

FIM Problem

Informally, pattern mining is described below.

Definition 2.1 (Pattern mining) *Pattern mining* is a set of problem instances, all of which aims at finding *combinatorial discrete patterns* from the given *transaction database* satisfying some *constraints*. A transaction database $\mathcal{D}$ is a set of transactions, which are *meaningful units* for representing data.

Pattern Mining

Transaction Database

Example 2.1 (Market Basket Analysis) A transaction $t \in \mathcal{D}$ is a set of *products bought by a consumer on some retailer*, and the database $\mathcal{D}$ stores all activities by consumers on a store. A fundamental technique of analyzing $\mathcal{D}$ is to discover *co-occurrence relationship* among products (e.g., some two products are often bought simultaneously even the two belong to completely different categories) to promote future sales. To capture such regularities behind databases, patterns are modeled as *itemset* (sets of products)[AIS93,AS94], and mining algorithms have been studied to efficiently find them.

Market Basket Analysis

Examples of patterns studied in the pattern mining literature are diverse; *association rules*[AS94], *negative rules*[YBYZ02], *sequential patterns*[AS95,PHMa+01,MR13], *periodical patterns*[HGY98,HDY99], *episodes*[MTIV97,Kat11], *graphs*[YWY03]. Several general patterns having specific problem settings are also studied. For example, *mining with utility*

values[HM07,TWSY10], *mining from streams*[GHP+03,CH08,HD09], *distributed cases*[CNFF96] are generalization for several application scenarios.

According to patterns, transactions are interpreted in various forms; a set of items bought by a user, a sequence of locations visited by a tourist, or a graph examined by an pharmacological experiment, and so on. In the following contexts, *patterns* are set to *mathematical discrete objects* consisting of elements in a *base-set*.

Base-set

Definition 2.2 (Base-set) Let $\mathcal{B}$ be a *base-set* for defining patterns.

For example to represent *itemsets*, a base-set $\mathcal{B}$ is set to be the set of all items that we want to analyze. As other examples, a set $\mathcal{B}'$ becomes a set of all symbols to represent sequences or strings, and another set $\mathcal{B}''$ should be a set of all labels representing some chemical atoms for modeling chemical compounds.

Pattern

Definition 2.3 (Patterns) A *pattern* is a *mathematical discrete object* consisting of *finite elements* from a pre-defined *base-set* $\mathcal{B}$.

For example, all of the followings are instances of patterns on $\mathcal{B}$.

- *itemsets* $X \subseteq \mathcal{B}$, sets of items in $\mathcal{B}$,
- *sequences* $s \in \mathcal{B}^+$, sequences consisting of elements in $\mathcal{B}$ where $\mathcal{B}^+$ denotes the *Kleene* closure of $\mathcal{B}$, and
- *graphs* $G = (V, E, \lambda)$ having elements in $\mathcal{B}$ as labels of vertices (or edges), i.e., V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and λ is a labeling function on V and E defined as $\lambda : V \cup E \to \mathcal{B}$.

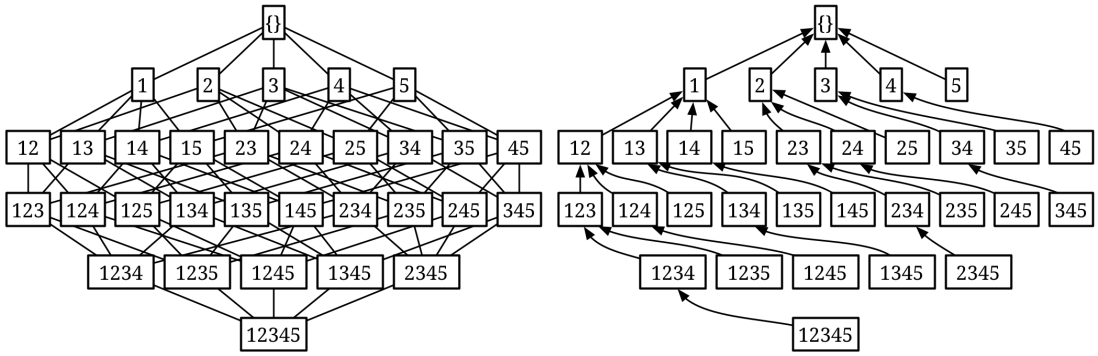
When the base-set $\mathcal{B}$ is fixed, the set of *all possible patterns* on $\mathcal{B}$ is decided according to patterns we are interested in. For example, if we are interested in itemsets on $\mathcal{B} = \{1, 2, 3, 4, 5\}$, we immediately can assume a search space of all itemsets as shown in Figure 2.1a. On the space of all itemsets, we need a *search strategy* that *efficiently traverse patterns* in the space in a *duplication-less manner*. For example, the way in Figure 2.1b is an example of traversing all itemsets in the space without duplications by assuming the order of items in $\mathcal{B} = \{1, 2, 3, 4, 5\}$, i.e., $1 \leq 2 \leq 3 \leq 4 \leq 5$ is assumed. The tree in Figure 2.1b is called a *enumeration tree*.

Enumeration Tree

Several formulations of patterns have been studied according to the purpose of pattern mining and a type of databases. A basic question in pattern mining is, as noted, to find all patterns satisfying some constraints, and clarify how difficult the problems are. The next section describes the most fundamental problem setting based on itemsets and *support* (or, *frequency*).

2.1. Frequent Itemset Mining

Let us recall the market basket analysis. As we would like to analyze consumers and their activities from databases, a base-set $\mathcal{B}$ is set to a set of all items (i.e., products). A pattern $X \subseteq \mathcal{B}$, called an itemset, representing the co-occurrence relationship



(a) All itemsets on $\mathcal{B} = \{1, 2, 3, 4, 5\}$.

(b) An enumeration tree on $\mathcal{B} = \{1, 2, 3, 4, 5\}$.

Figure 2.1.: The search space of itemsets and an enumeration tree on $\mathcal{B} = \{1, 2, 3, 4, 5\}$.

among items. For the convenience of notations we could write an itemset $\{i, j, k\}$ as ijk briefly, and a singleton itemset $\{i\}$ as i for short. A *transaction* is defined as a pair $t = \langle tid, X \rangle$, where $tid \in \mathbb{N}$ is a transaction identifier and $X \subseteq \mathcal{B}$ is an itemset. Without loss of generality we shall represent tid and X of $t = \langle tid, X \rangle$ as $tid = t.tid$ and $X = t.X$, respectively. For the convenience of denoting set operations, we sometime ignore the tid of a transaction t. For example, given a transaction $t = \langle tid, X \rangle$ and an itemset Y, $t \subseteq Y$ means $X \subseteq Y$ by ignoring $t.tid$. Now a formulation of transaction databases is a set of transactions: $\mathcal{D} = \{\langle tid, X \rangle \mid tid \in \mathbb{N}, X \subseteq \mathcal{B}\}$.

When we focus on the co-occurrence relation among items, a basic assumption is that if two items occur in common frequently, the two are meaningful. This simple assumption to evaluate an itemset is implemented in the form of *support* (or *frequency*). A transaction t is said to *contain*, *support*, or *cover* an itemset Y, denoted by $X \subseteq t$, if all items $i \in Y$ it holds that $i \in t.X$ (i.e., $Y \in t.X$). The *cover* of an itemset is defined as the set of identifiers of transactions that contain the itemset:

Support
Cover

$$\text{cover}(X, \mathcal{D}) = \{t.tid \mid t \in \mathcal{D} \text{ s.t. } X \subseteq t\}.$$

If the cardinality of the cover of an itemset X is large, we can say that items from X occur in $\mathcal{D}$ frequently in common, and this is also applicable to evaluate the co-occurrence relation of items. The *support* of an itemset X in $\mathcal{D}$ is, based on this observation, defined as the number of transactions in $\mathcal{D}$ covering X:

$$\text{supp}(X, \mathcal{D}) = |\text{cover}(X, \mathcal{D})| = |\{t \in \mathcal{D} \mid X \subseteq t\}|.$$

For the convenience, the *frequency* of an itemset X in $\mathcal{D}$ is introduced as the *relative* number of transactions covering X: $\text{freq}(X, \mathcal{D}) = \frac{\text{supp}(X, \mathcal{D})}{|\mathcal{D}|}$.

Problem 2.1 (The Frequent Itemset Mining problem) The *frequent itemset mining* (FIM) problem is to find the collection $\mathcal{F}$ out of all possible itemsets $2^{\mathcal{B}}$ *whose support is higher than a user-defined threshold.* The collection $\mathcal{F}$ is defined as

FIM Problem

$$\mathcal{F} = \{X \subseteq \mathcal{B} \mid \text{supp}(X, \mathcal{D}) \geq \theta\} = \{X \subseteq \mathcal{B} \mid \text{freq}(X, \mathcal{D}) \geq \sigma\},$$

where $\theta \in (0, |\mathcal{D}|]$ and $\sigma \in (0, 1]$. The itemsets in $\mathcal{F}$ are called *frequent* (or, *frequent itemsets*). The user-given parameters θ (or, σ) are set according to problems or users' interest on analyzing $\mathcal{D}$.

In the following, without loss of generality, we deal with $\theta \in (0, |\mathcal{D}|]$; the absolute number of the support of an itemset X.

Association Rule

Based on frequent itemsets computed in the FIM problem, *association rules* have been studied[AS94]: For itemsets X and Y, an association rule between X and Y is denoted as $X \to Y$. We interpret an association rule $X \to Y$ as a pattern representing *implication*: "If X occurs in $\mathcal{D}$, then with *high probability* Y follows it". To evaluate whether or not such a rule is interesting, the *confidence* of an association rule $X \to Y$ is defined as:

Implication
Confidence

$$\text{conf}(X \to Y, \mathcal{D}) = \frac{\text{supp}(X \cup Y, \mathcal{D})}{\text{supp}(X, \mathcal{D})}.$$

Problem 2.2 (The Association Rule Mining problem) The *association rule mining* problem is to list all association rules whose confidence is higher than a user-defined threshold τ, and for which the union of X and Y (in the rule of the form $X \to Y$) is frequent under the constraint $X \cap Y = \emptyset$.

For association rule mining, methods for the above frequent itemset mining problem can be applicable[AS94,ZPOL97].

2.1.1. Foundations of Mining Algorithms

Not only for itemsets but also for various patterns, to find the set $\mathcal{F}$, mining algorithms follow a common strategy known as *Generation-and-Test*. Recall the search space illustrated in Figure 2.1a. Now the set $\mathcal{F}$ of pattern mining results can correspond to a subspace of the whole space. To find such a space based on *Generation-and-Test*, the process consists of two phases:

Generation-and-Test

Generation Phase Algorithms try to generate all possible solutions.

Test Phase Algorithms test to see if a possible solution is expected.

In the Generation Phase, we had better avoid the duplications of generating possible solutions. To ensure the issue, a typical search strategy, e.g., *Breadth-First Search* (BFS) or *Depth-First Search* (DFS) strategies, can be adopted with checking the duplications in pattern mining. Below we review three algorithms in both search strategies for itemsets.

Apriori

Apriori Algorithm The APRIORI algorithm[AS94] is a fundamental algorithm using the BFS strategy to generate and test all frequent itemsets. The fundamental property adopted is often referred as *anti-monotonicity*.

Algorithm 1 The APRIORI algorithm[AS94,AH14]

Input: A transaction database $\mathcal{D}$ and a threshold θ

Call APRIORI ($\mathcal{D}, \theta$)

1: **procedure** APRIORI (a transaction database $\mathcal{D}$, a threshold θ)
2: Generate frequent singleton itemset, denoted by $\mathcal{F}_1$
3: $l = 1$
4: **while** $\mathcal{F}_l$ is not empty **do**
5: Generate $\mathcal{C}_{l+1}$ by using joins on $\mathcal{F}_l$
6: Prune $\mathcal{C}_{l+1}$ with the subset pruning trick
7: Generate $\mathcal{F}_{l+1}$ by counting candidates $\mathcal{C}_{l+1}$ with respect to $\mathcal{D}$ and θ
8: $l \leftarrow l + 1$

Definition 2.4 (anti-monotonicity on the FIM problem) For itemsets X and Y on a base-set $\mathcal{B}$ and a database $\mathcal{D}$, $X \subseteq Y$ implies $\text{supp}(X, \mathcal{D}) \geq \text{supp}(Y, \mathcal{D})$.

This property on the itemsets and the support immediately arises the following corollary, and this fact is the main concept used in the APRIORI algorithm to find all frequent itemsets in the FIM problem.

Corollary 2.1 (Traversing itemsets with anti-monotonicity) For $X \in 2^{\mathcal{B}}$, if there exists a subset $X' \subset X$ such that $|X'| = |X| - 1$ and X' is an infrequent, the itemset X should become infrequent as well.

The search space is formally formulated by a lattice structure (For lattices, see appendix A) on the set $2^{\mathcal{B}}$ of all itemsets, as we saw in Figure 2.1a. Following Corollary 2.1, the basic strategy of the traverse is to *join* two frequent itemset X_1 and X_2 of size l to make a new possible frequent itemset of size $l + 1$. After getting the candidates, we filter them by the property in Corollary 2.1, and then we test whether or not the new candidate itemset is frequent, by accessing to the database $\mathcal{D}$. In the APRIORI algorithm, the traverse of the itemset lattice begins from the level 1, i.e., the *singleton* itemsets. If an item $i \in \mathcal{B}$ is infrequent, any itemsets containing this item i are no longer frequent, and then the search space corresponding this item i should be pruned (corresponding to the gray sub-lattice containing the item 4, for example in Figure 2.2). In summary, on some level l, i.e., when considering computing frequent itemsets of size l, we do the followings:

Generation Phase We generate all candidates in C_{l+1} on level $l + 1$ by using *only* frequent itemsets of size l from the set $\mathcal{F}_l$. Note that a possible itemset of size $l + 1$ contains no infrequent itemsets of size l, and this property is checked.

Test Phase For all $X \in C_{l+1}$, we access the database $\mathcal{D}$ to evaluate the measure $\text{supp}(X, \mathcal{D})$, and if X is now frequent, this set X should be included in $\mathcal{F}_{l+1}$.

The process halts when the set $\mathcal{F}_l$ on some level l is empty. The overall process of APRIORI is in Algorithm 1.

TID	Itemset
100	{1, 3, 4}
200	{2, 3, 5}
300	{1, 2, 3, 5}
400	{2, 5}

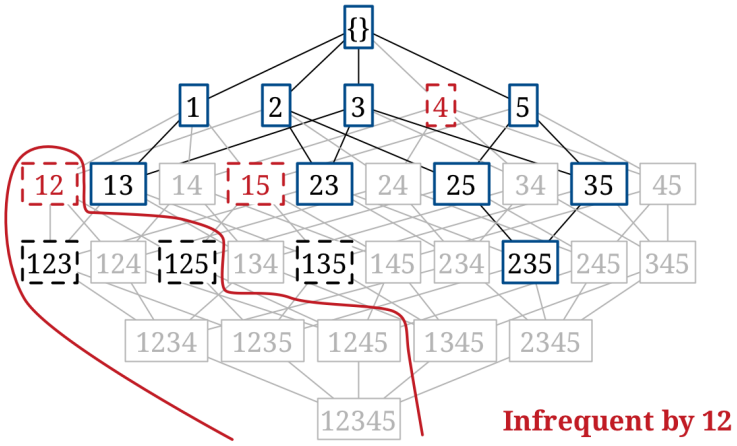


Figure 2.2.: Three examples from Example 2.2. (1) Once the item 4 is turned to be infrequent, the part containing 4 is not needed to check. (2) The itemset 12 is infrequent and consequently the following parts $123, 145, \cdots$ are not taken into account. (3) Though the itemset 123 can be made by the join of 13 and 23, it contains the infrequent itemset 12 and consequently the itemset 123 is not needed to check.

Example 2.2 (A FIM instance) Let $\mathcal{B} = \{1, 2, 3, 4, 5\}$, $\theta = 2$ (i.e., $\sigma = 0.4$), and $\mathcal{D} = \{\langle 100, 134\rangle, \langle 200, 235\rangle, \langle 300, 1235\rangle, \langle 400, 25\rangle\}$.

1. The item 4 is infrequent, and $\mathcal{F}_1 = \{1, 2, 3, 5\}$ is computed.
2. The set C_2 is computed from $\mathcal{F}_1$ as $C_2 = \{12, 13, 15, 23, 25, 35\}$. Out of items in C_2, it turns out that $13, 23, 25$, and 35 are frequent in $\mathcal{D}$. Thus the set $\mathcal{F}_2$ is now $\mathcal{F}_2 = \{13, 23, 25, 35\}$.
3. The set C_3 is now computed as $C_3 = \{235\}$, and it is also done $\mathcal{F}_3 = \{234\}$. The set C_4 is now empty and finished.

The key generation is in step 3: itemsets $123, 145$, and 135 *should not be included in* C_3 because all of them contain infrequent itemsets of size 2. The result set is $\mathcal{F} = \{1, 2, 3, 5, 13, 23, 25, 35, 235\}$. See also Figure 2.2 of these operations.

Apriori-based algorithms have been studied not only for itemsets but also several patterns. As the fundamental idea in the Apriori algorithm is simple, consisting of two phases (the Generation Phase and the Test Phase), we can construct similar search methods to various patterns. This point will be explained more in Section 2.3.

A drawback of the Apriori algorithm is that it requires to generate candidate patterns C_{l+1} from $\mathcal{F}_l$. This process, in practice, requires a large memory space and computation resources, and in some cases the computation is practically impossible. In addition, the Test Phase is also hard if the database $\mathcal{D}$ gets large and when the measure gets hard to compute in patterns. That is, 1) the candidate generation, 2) the search space of patterns, and 3) the evaluation of the measure $\mathrm{freq}(\cdot)$ are common problems in pattern mining. Because algorithms need to generate combinations of frequent itemsets it should be time consuming to test $\frac{l(l+1)}{2}$ ways in the naïve manner. An efficient implementation of used is that in Figure 2.1b,

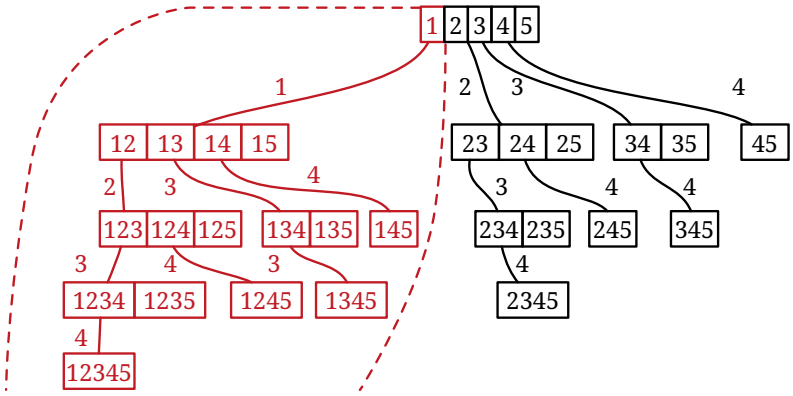


Figure 2.3.: A prefix tree on $\mathcal{B} = \{1, 2, 3, 4, 5\}$, where the total order on integers are used to remove redundant candidate generations, i.e., the canonical parents are used to build the tree without duplications. Common prefixes are labeled on edges. The red subtree shows the parts related to the item 1.

where algorithms adopt the *canonical representation of itemsets* by assuming the order on items in $\mathcal{B}$, which enable us to avoid reach the same itemset $X \subseteq \mathcal{B}$ twice in traversing itemsets. From the viewpoint of enumeration, the canonical representation supports the canonical parent, denoted by $\mathit{parent}_{\mathrm{c}}(I)$, of the itemset X by $\mathit{parent}_{\mathrm{c}}(X) = X \setminus \{\max_{i \in X} i\}$. This discussion is related to the *Reverse Search* method in enumeration[AF96]. In addition, the canonical representation provides an enumeration tree in the form of *prefix trees* as shown in Figure 2.3. Reverse Search

To overcome issues in algorithms adopting the BFS strategy, algorithms based on the DFS strategy and those using compressed data structures had attracted much attention. We explain the Eclat algorithm and the FP-Growth algorithm below.

Eclat The Eclat algorithm[ZPOL97] checked itemsets in the *lexicographical order* (See Figure 2.3), which adopts the vertical representation of databases. Eclat

> **Example 2.3 (Cont'd from Example 2.2)** Recall that we have a database $\mathcal{D} = \{\langle 100, 134\rangle, \langle 200, 235\rangle, \langle 300, 1235\rangle, \langle 400, 25\rangle\}$. It is called a *horizontal representation*. On the other hand, *vertical representations* stores the inverse index of the occurrences of items. For example, the item 1 now occurs at transactions of id 100 and 300. Therefore, the vertical representation of the item 1 is the list $\langle 100, 300\rangle$ and its bit array is given as 1010.

Let us consider the *divide-and-conquer* strategy for the FIM problem: We divide the database $\mathcal{D}$ into two sub-databases $\mathcal{D}_1$ and $\mathcal{D}_2$, where $\mathcal{D}_1$ satisfies some condition and $\mathcal{D}_2$ does not satisfy it. For example, focusing on the item 1 on the prefix tree in Figure 2.3, where we can set $\mathcal{D}_1$ to be a database $\mathcal{D}$ consisting of transactions having the item 1, and $\mathcal{D}_2$ to be that of transactions which do not contain the item 1. This divide-and-conquer strategy can be interpreted with the DFS strategy on the prefix tree. That is, the algorithm first tries to solve the problem for the item 1, and then it processes the problem for the itemsets 12 in the DFS manner. In the Eclat algorithms, such partitions are implemented by a vertical representation. Such a Divide and Conquer

Algorithm 2 The ECLAT algorithm [ZPOL97]

Input: A transaction database $\mathcal{D}$ and a threshold θ

Let F be a global variable for $\mathcal{F}$. Call ECLAT ($\mathcal{D}, \theta$).

1: **procedure** ECLAT (a transaction database $\mathcal{D}$, a threshold θ)
2: $F = \{(\emptyset, |\mathcal{D}|)\}$ ▷ Global variable
3: $C_\emptyset = \{(x, \mathcal{D}_x) \mid x \in \mathcal{B}, \mathcal{D}_x = \{t \in \mathcal{D} \mid x \in d\}\}$ ▷ The 1st level
4: $C'_\emptyset = \text{freq}(C_0) \equiv \{(x, \mathcal{D}_x) \in C_0 \mid |\mathcal{D}_x| \geq \theta\}$.
5: Call TRAVERSE$(\emptyset, C'_\emptyset)$
6: **procedure** TRAVERSE(a prefix p, the conditional database C)
7: **for** $(x, \mathcal{D}_x) \in C$ **do**
8: $q = p \cup \{x\}$
9: $C_q = \{(y, \mathcal{D}_x \cap \mathcal{D}_y) \mid (y, \mathcal{D}_y) \in C, y > x\}$.
10: $C'_q = \text{freq}(C_q) \equiv \{(y, \mathcal{D}_y) \in C_q \mid |\mathcal{D}_y| \geq \theta\}$.
11: **if** $C'_q \neq \emptyset$ **then**
12: TRAVERSE (q, C'_q) ▷ Depth-First Search from level k to $k+1$
13: $F = F \cup \{(q, |\mathcal{D}_x|)\}$

Conditional Database

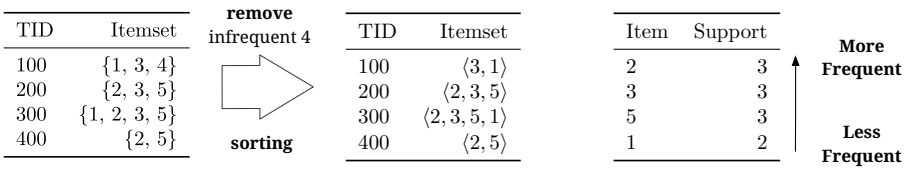
sub-database partitioned with respect to the itemset given is called a *conditional database*. The pseudo code of the ECLAT algorithm is given in Algorithm 2, where conditional databases on the itemset q (also called *prefix*) is represented by $\mathcal{D}_x$, the bit matrix (a collection of bit arrays) representation is used to compute conditional databases.

FP-growth

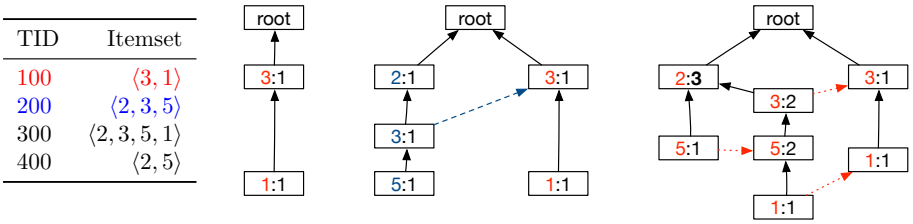
FP-Growth The FP-GROWTH algorithm [HPY00,HPYM04] is an algorithm based on the DFS strategy using *compressed data structures*. The key strategy of FP-GROWTH is traversing the database $\mathcal{D}$ first and store the information required to evaluate the support on a *trie-like tree structure*, called *FP-tree*, to minimize the number of accesses to the database $\mathcal{D}$ required when evaluating the support. In pre-processing of FP-GROWTH, the algorithm filters out infrequent itemsets. We then check items in support descending and lexicographically descending order as shown in Figure 2.4a.

Next, the algorithm scan the filtered database once, and store lexicographically ordered items in a trie, named *FP-tree*. The process is illustrated in Figure 2.4b. To traverse FP-trees efficiently, they should have links between the same items (e.g., the link between the box 3:1 to the same box in Figure 2.4b).

After building an FP-tree, we traverse the tree, as its *Generation Phase*, from the bottom of the *item table* in which frequent items are stored; The algorithm traverses items in support ascending order. When visiting the parent from the item, the support value is transferred from the leaf to the root of the FP-tree (e.g., when the item 1 of support 1 is accessed, the supports of the items $2, 3, 5$ are now revised with the support 1 of the item 1). This part corresponds to the part getting conditional databases in the ECLAT algorithm.



(a) A pre-processing for the FP-growth algorithm.



(b) Building an FP-tree; the first transaction $\langle 3, 1\rangle$ is stored directly with the support, the second transaction $\langle 2, 3, 5\rangle$ is inserted and the *internal link* (dashed arrow) from 3 in the second transaction to the first transaction is set, and finally the FP-tree is given after reading all transactions.

Figure 2.4.: The summary of the construction of FP-trees.

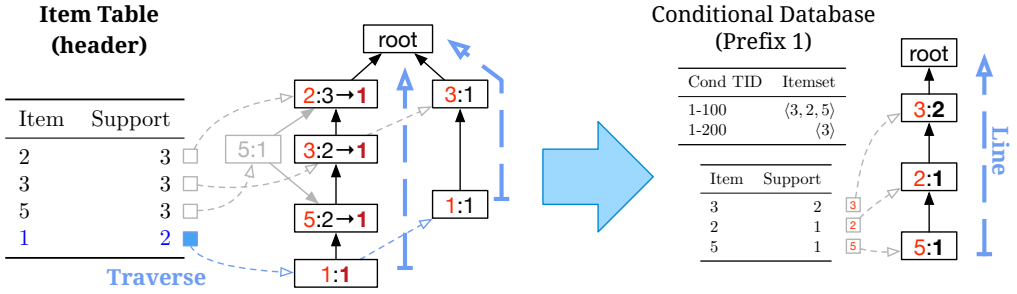


Figure 2.5.: A conditional database with prefix $\{1\}$, where the values above the node 1 are updated, and this leads us to compute the conditional database with the prefix 1. If the FP-tree is like a *line*, combinations of items; itemsets 12, 13, 15, etc. are generated and tested.

Example 2.4 (Traversing FP-trees) In focusing on the item 1 (e.g., the conditional database in Figure 2.5), the support value 1 is adopted to the ancestors 235, and values stored with 235 are updated with 1. From the updated sub-FP-tree, the conditional database with prefix 1 is computed. In the process, we can confirm that with the item 1, the item 3 occurs twice, and the items 2 and 5 occur once, consequently have the answer that 13 has the support 2 and both 12 and 15 have the value 1, respectively, as the Test phase. This process is possible when the FP-tree is a line as shown in Figure 2.5. If the database contains further items (i.e., the obtained sub-FP-tree is not a line), the algorithm again compute the projected database to traverse the search space more deeply.

The step above is summarized in Algorithm 3.

Let us see an (small) experimental evaluations of the three algorithms above. We adopted two datasets mushroom and connect, and applied three algorithms with varying the threshold parameter, and measured the times required for listing all frequent itemsets. Generally speaking, algorithms based on the BFS strategy

Algorithm 3 The FP-tree construction / FP-growth algorithm[HPY00,HPYM04]

Input: A transaction database $\mathcal{D}$ and a threshold θ

Let T =FP-TREE-CONSTRUCTION ($\mathcal{D}, \theta$). Call FP-GROWTH ($T, \emptyset$).

procedure FP-TREE-CONSTRUCTION (a transaction database $\mathcal{D}$, a threshold θ)
  Scan $\mathcal{D}$ once and collect $\mathcal{F}_1$ with its support
  Sort $\mathcal{F}_1$ in the support ascending order as $FList$
  Create the root of a FP-tree T
  for each transaction $t \in \mathcal{D}$ **do**
    Remove infrequent items in t and sort it
    Let SL_t be a sorted list of frequent items in t ▷ the list $SL_t = \langle h \mid H \rangle$
    Call INSERT ($\langle h \mid H \rangle$); a function inserting SL_t in the tree T
  return T

procedure FP-GROWTH (a FP-tree T, a prefix α)
  Let SL be a sorted frequent itemset. In a support-ascending order of items in SL, do the following:
  for each item $i \in SL$ **do**
    Take the pointer list of nodes related to the item i
    Consider the parent of i on T
    Compute the conditional database *CDB*
    Build the conditional FP-tree T' with the item F'
    if The conditional FP-tree T' is a line form **then**
      Compute combinations of itemsets with α
      Compute its support and **output** if frequent
    else
      Continue the procedure with FP-GROWTH($CT, \alpha \cup \{i\}$)

are slower than those using the DFS strategy. Similar results are also obtained for other patterns; e.g. sequential patterns by comparing apriori-based methods and projection-based methods.

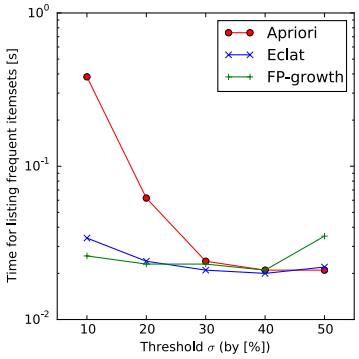
Figure 2.6 illustrates times required to enumerate frequent itemsets with the three algorithms above from two benchmark datasets; mushroom and connect. As results indicate, the DFS-based methods are much faster than those using the BFS-strategy.

2.2. Reducing Outputs

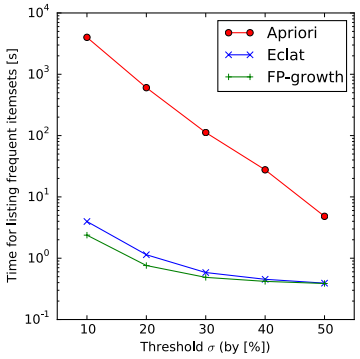
Although some pattern mining instances are tractable (e.g., itemsets mining), results of mining algorithms are sometimes troublesome. That is, even for the FIM problem, the time complexity of computing the set $\mathcal{F}$ is exponential to the input parameter (e.g., the size of the input database $|\mathcal{D}|$) in the worst. This drawback, the fact that the number of enumerated patterns is in general extremely large and it is hard to utilize the set $\mathcal{F}$ in practice, is known as *pattern combinatorial explosions*. This is undesirable for both theoretical and practical aspects.

Combinatorial Explosion

From the theoretical viewpoint, the time complexity of almost all mining algorithms cannot be bound by a polynomial of the input parameter. To discuss the



(a) For mushroom, times for listing frequent itemsets with varying with varying $\sigma = 50\%, 40\%, \ldots, 10\%$.



(b) For connect, times for listing frequent itemsets with varying with varying $\sigma = 50\%, 40\%, \ldots, 10\%$.

Figure 2.6.: Times for enumerating all frequent itemsets for two datasets (mushroom, connect) with three algorithms explained above, where the y-axes are logarithmic scale.

theoretical point, we introduced *listing complexity* in Section 2.3 and this complexity is a main subject in Chapter 3. Listing Complexity

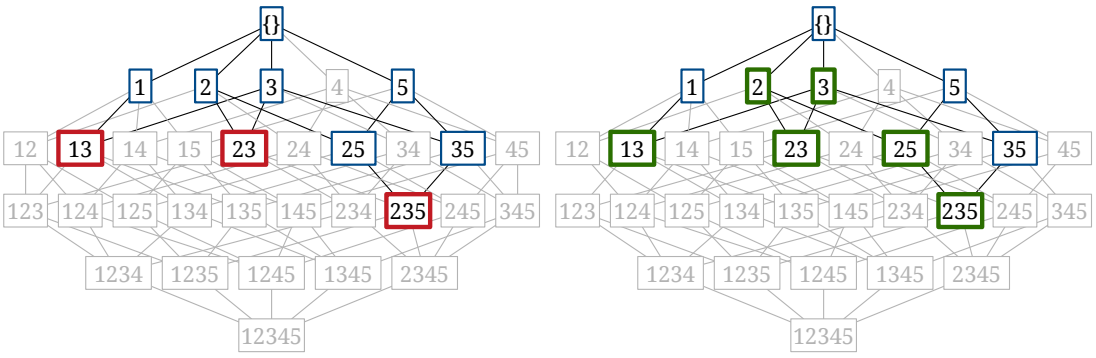
From the practical viewpoint, we should decide which enumerated patterns in the set $\mathcal{F}$ are really important for some Knowledge Discovery tasks and decision making problems. The following chapters focuses on developing structures which support users to search for the set $\mathcal{F}$ with features by focusing on post-processing in Chapter 4 and Chapter 5 of pattern mining. Compared with pre-processing, post-processing of pattern mining and evaluation methods have been studied for a long time. However, for large scale data, more advances are now required. That is why behaviors of mining algorithms are often extreme, i.e., results are heavily sensitive with respect to the threshold parameter. This phenomenon arises many challenges in post-processing. Therefore, in Chapter 6 we deal with pre-processing of pattern mining so that we can control enumeration algorithms generating. In summary, we simultaneously investigate both pre-processing and post-processing in the thesis to overcome the issues from practical viewpoints.

2.2.1. Maximal and Closed Itemsets

An idea to reduce the number of frequent itemsets is to make a cluster of frequent itemsets with constraints. Two representative ways are computing *maximal* and *closed itemsets*.

Definition 2.5 (Maximal Itemset[Bay98]) An itemset X is *maximal* if and only if none of its proper supersets are frequent. Such an itemset X is called a maximal itemset. The set $\mathcal{MF}$ denotes the set of all maximal itemsets. Maximal Itemset

Maximal itemsets support us to recover every frequent itemsets $\mathcal{F}$ *without* its supports. On the other hand, closed itemsets support us to recover every frequent itemsets $\mathcal{F}$ *with* their supports.



(a) The lattice with maximal itemsets indicated with red rectangles.

(b) The lattice with closed itemsets indicated with green rectangles.

Figure 2.7.: Examples of maximal itemsets (red) and closed itemsets (green) from the set $\mathcal{F} = \{1, 2, 3, 5, 13, 23, 25, 35, 235\}$ of frequent itemsets (blue).

Closed Itemset

Definition 2.6 (Closed Itemset[PBTL99]) An itemset X is *closed* if and only if none of its proper supersets have the same support of X. Such an itemset X is called a closed itemset. The set $\mathcal{CF}$ denotes the set of all closed itemsets.

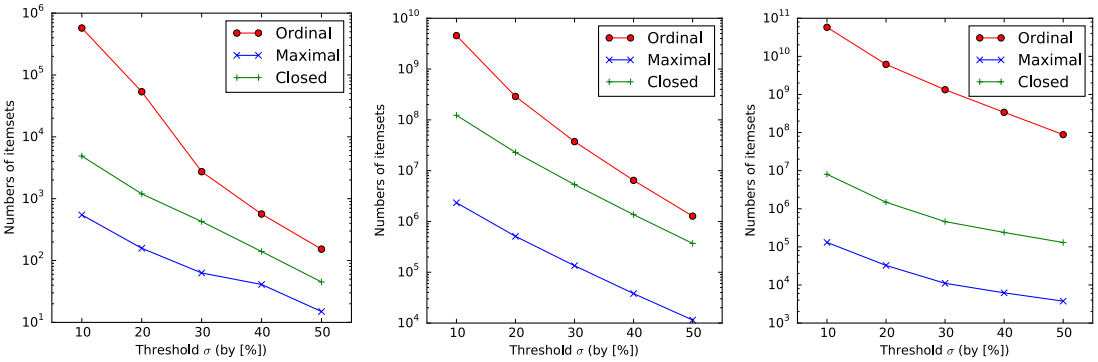
Because we can recover the support of each frequent itemset from the set of all closed itemsets, we can regard closed itemsets as *loss-less, compact representations of frequent itemsets*.

Example 2.5 (Maximal and closed itemsets) Recall the lattice we used in Figure 2.1a and Example 2.2, where the set $\mathcal{F}$ of frequent itemsets with the parameter $\theta = 2$ is $\{1, 2, 3, 5, 13, 23, 25, 35, 235\}$. In the example, $\mathcal{MF} = \{13, 23, 235\}$ and $\mathcal{CF} = \{2, 3, 13, 23, 25, 235\}$, and it holds that $\mathcal{MF} \subseteq \mathcal{CF} \subseteq \mathcal{F}$. For example in maximal itemsets, from the frequent itemset 235 of support 2, we can recover frequent itemsets 5, 25, and 35, but the supports of them cannot be recovered. On the other hand in closed itemsets, from the closed itemset 13, the itemset 1 is recovered with its support (the same to the support of 13), and the support of 5 is the same to 25, and that of 35 is also the same to 235. That is, we can recover the supports of all frequent itemsets from closed itemsets. See Figure 2.7 of the maximal and closed itemsets, continued from Example 2.2.

From experiments shown in Figure 1.5 (numbers of frequent itemsets) and Figure 2.6 (times for listing), we here confirm that the relation $\mathcal{MF} \subseteq \mathcal{CF} \subseteq \mathcal{F}$ holds for the selected benchmark datasets. Figure 2.8 showed times required to list all ordinal, maximal, and closed itemsets from the selected datasets; mushroom in Figure 2.8a, chess in Figure 2.8b, and connect in Figure 2.8c. Compared with maximal itemsets, numbers of closed itemsets are a bit larger, but they are still smaller than those of ordinal frequent itemsets, and recall that we can recover all information from closed ones. In contrast, with respect to running time of mining algorithms, the closed itemsets are more time-consuming than others (See Figure 2.9; results with mushroom in Figure 2.9a, chess in Figure 2.9b, and connect Figure 2.9c).

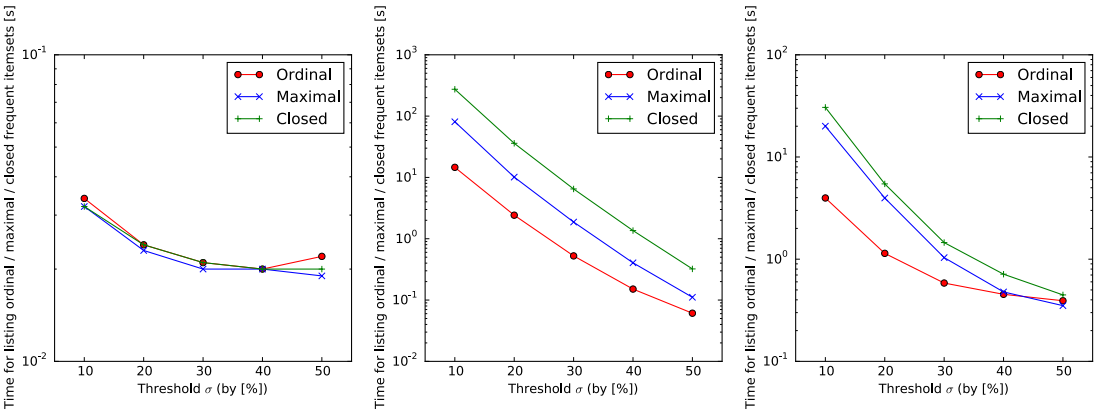
Closure
Formal Concept Analysis

The closed itemsets are also important from the viewpoint of *closures*, which is a key conception used in *Formal Concept Analysis*, which is followed and explained



(a) mushroom (b) chess (c) connect

Figure 2.8.: Numbers of ordinal, maximal, and closed itemsets from three datasets with varying the threshold parameter, where the y-axis is the logarithmic scale.



(a) mushroom (b) chess (c) connect

Figure 2.9.: Times of listing ordinal, maximal, and closed itemsets from three datasets with varying the threshold parameter, where the y-axis is the logarithmic scale.

more formally in Chapter 4, where the main discussion is the generalization of closures for structured patterns beyond itemsets (i.e., binary data), and in Chapter 5, where the problem of selecting the subset of the whole set $\mathcal{F}$ based on a Information Theory-based criterion.

2.2.2. Pattern Selection and Mining Sets of Patterns

For frequent $\mathcal{F}$, maximal $\mathcal{MF}$, and closed itemsets $\mathcal{CF}$, we know the relation $\mathcal{MF} \subseteq \mathcal{CF} \subseteq \mathcal{F}$. More generally, towards the purpose of reducing the output size $|\mathcal{F}|$, a straightforward problem setting is selecting a *subset* $M \subseteq \mathcal{F}$ based on some measure. The maximal itemsets and closed itemsets can be regarded as special cases of selecting the subset of the whole set $\mathcal{F}$ of patterns enumerated, in the *evaluation / post-processing part* of the KDD process.

Putting more constraints than maximal or closed patterns is a fundamental approach (e.g., Chapter 7 in [AH14]). The anti-monotonicity property as we used in itemsets is important to achieve efficient pattern mining algorithms [MT97,HLN99]. Top-k

Top-k Mining

mining is an example of problems adopting more restrict constraints; For example in [HWLT02], Han et al. formalized the problem to find *top-k closed itemsets of minimal length min_l*: finding only a closed itemset X satisfying there exists no more than $(k-1)$ closed itemsets of length at least min_l whose support is higher than that of X. Intuitively, this is a problem to find relatively long, frequent, and rare closed itemsets out of the set $\mathcal{CF}$ of all closed itemsets.

Pattern Set Mining

Generally speaking, problems getting a set $S \subseteq \mathcal{F}$ from the whole set of frequent patterns are referred as *Pattern Set Mining*, consisting of two problem settings; *Supervised* and *unsupervised* pattern set mining. This thesis only discusses pattern mining settings in the *unsupervised* manner, where we select such a subset S based on some criteria or models. This is the problem dealt in Chapter 5.

2.3. Theory Extraction Framework

As many pattern mining problems are heavily depending on the implicit order of the base-set $\mathcal{B}$ and a partial order among patterns (e.g., the subset relation among itemsets) to avoid duplications, a formal description of pattern mining instances is characterized with such an order. Mannila et al. gave a high-level description of pattern mining instances as the *Theory Extraction Problem*.

Theory Extraction Problem

Definition 2.7 (Theory Extraction Problem [MT97]) Let us consider the 4-tuple $(\mathcal{D}, \mathcal{L}, \preccurlyeq, q_D)$ consisting of 1) a transaction database $\mathcal{D}$, 2) a pattern language $\mathcal{L}$, expressing properties or defining subgroups of the data in which elements of $\mathcal{L}$ are referred to as *sentences* or *patterns*, 3) a partial order $\preccurlyeq$ on $\mathcal{L}$, representing *generalization* of patterns, and 4) an interestingness measure predicate $q_D : \mathcal{L} \to \{\text{true}, \text{false}\}$ to evaluate whether or not a pattern $P \in \mathcal{L}$ is *interesting* on $\mathcal{D}$. The task, computing the set

$$Th(\mathcal{L}, \mathcal{D}, q_D) = \{P \in \mathcal{L} \mid q_D(P) \text{ is true}\},$$

is called a *Theory Extraction Problem* on $(\mathcal{D}, \mathcal{L}, \preccurlyeq, q_\mathcal{D})$. The set $Th(\mathcal{L}, \mathcal{D}, q_D)$ of *interesting patterns* is called a *theory* of $\mathcal{D}$ with respect to $\mathcal{L}$ and $q_\mathcal{D}$.

Theory

In many practical pattern mining instances, a partial order $\preccurlyeq$ on $\mathcal{L}$ is assumed. For two patterns P_1 and P_2 in $\mathcal{L}$, we say that P_1 is more general than P_2, denoted by $P_1 \preccurlyeq P_2$ by the partial order $\preccurlyeq$. In addition, in many cases we often assume that $\mathcal{L}$ has a *least element* with respect to $\preccurlyeq$. With a partial order $\preccurlyeq$, a general form of anti-monotonicity is now given.

Definition 2.8 (Anti-monotonicity) Let $\mathcal{L}$ and $\preccurlyeq$ be a pattern language and a natural partial order. An interestingness measure $q_\mathcal{D}$ *anti-monotone* with respect to $\preccurlyeq$ if for two pattern P_1, P_2 such that $P_1 \preccurlyeq P_2$, it holds that $q_\mathcal{D}(P_1) = \text{true}$ implies $q_\mathcal{D}(P_2) = \text{true}$.

Levelwise Search

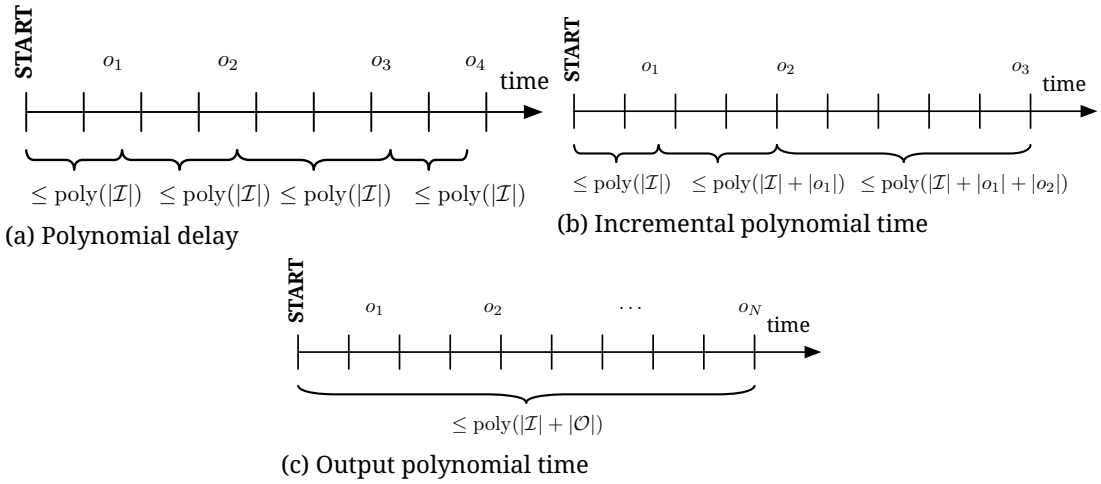
Mannila et al. in [MT97] also described a formal algorithm, named *levelwise search algorithm* as shown in Algorithm 4. In Line 1, we begin the enumeration from the

Algorithm 4 The Levelwise Search algorithm [MT97]

Input: A 4-tuple $(\mathcal{D}, \mathcal{L}, \preccurlyeq, q_D)$.

Call LevelwiseSearch $(\mathcal{D}, \mathcal{L}, \preccurlyeq, q_D)$.

1: **procedure** LevelwiseSearch (an 4-tuple $(\mathcal{D}, \mathcal{L}, \preccurlyeq, q_D)$)
2: $C_1 = \{P \in \mathcal{L} \mid$ there is no $P' \in \mathcal{L}$ such that $P' \prec P\}$.
3: $l \leftarrow 1$
4: **while** $C_l \neq \emptyset$ **do**
5: $\mathcal{F}_l = \{P \in C_l \mid q_D(P)$ is true$\}$
6: $C_{l+1} = \{P \in \mathcal{L} \mid$ for all $P' \prec P$ we have $P' \in \bigcup_{j \leq l} \mathcal{F}_j\} \setminus \bigcup_{j \leq l} C_j$
7: $l \leftarrow l + 1$
8: **output** $\bigcup_{j<l} \mathcal{F}_j$.



(a) Polynomial delay

(b) Incremental polynomial time

(c) Output polynomial time

Figure 2.10.: Three classes of listing complexity, from the input $\mathcal{I}$ to the output list $\mathcal{O} = \langle o_1, \ldots, o_N \rangle$.

minimal elements on the poset $(L, \prec)$, and in Line 5 the candidates in C_l is checked and filtered to get $\mathcal{F}_i$. Using the combination of only frequent patterns before (corresponding to $\bigcup_{j \leq l} \mathcal{F}_j$), without duplications (corresponding to $\setminus \bigcup_{j \leq l} C_j$), build the set of next candidates C_{l+1} in Line 6. This process is iterated til we cannot get more candidates in the iteration.

Enumeration
Listing

Compared with typical algorithmic problems, the *enumeration*[*] problems or *listing* problems are a little different from them on the viewpoint of computational complexity. That is, a common feature of many listing problems is that the size of the output can be exponential in that of the input. It is obvious for such cases that we cannot have algorithms that enumerate the output of a problem instance in time polynomial in the size of the input. This is the motivation of taking into account the size of the output in evaluating listing algorithms. The following listing complexity classes are usually distinguished in the literature (see, e.g., [JYP88], the web

[*] An enumeration is an ordered listing of all the items in a collection. Usually, the items in the list satisfy some conditions in common. In this sense, pattern mining instances can be solved by efficient enumeration algorithms.

page[†]): For some input $\mathcal{I}$, let $\mathcal{O}$ be the output set of some finite cardinality N. Then the elements of $\mathcal{O} = \langle o_1, \ldots, o_N \rangle$, are listed with

Polynomial delay if the time before printing o_1, the time between printing o_i and o_{i+1} for every $i = 1, \ldots, N-1$, and the time between printing o_N and the termination is bounded by a polynomial of the size of $\mathcal{I}$,

Incremental polynomial time if o_1 is printed with polynomial delay, the time between printing o_i and o_{i+1} for every $i = 1, \ldots, N-1$ (resp. the time between printing o_N and the termination) is bounded by a polynomial of the combined size of $\mathcal{I}$ and the set $\{o_1, \ldots, o_i\}$ (resp. $\mathcal{O}$),

Output polynomial time (or polynomial total time) if $\mathcal{O}$ is printed in time polynomial in the combined size of $\mathcal{I}$ and the *entire* output $\mathcal{O}$.

In the following poly(n) means some polynomials with the parameter n. Figure 2.10 illustrates these three types. Clearly, polynomial delay implies incremental polynomial time, which, in turn, means output polynomial time. Furthermore, in contrast to incremental polynomial time, the delay of an output polynomial time algorithm may be exponential in the size of the input even before printing the first element of the output.

2.4. References

Han et al.[HCXY07] surveyed pattern mining thoroughly and gave many links and overviews of the field and several algorithms. For graphs, Washio and Motoda[WM03] surveyed the field. A recent book for such survey is by Aggarwal and Han[AH14].

Itemsets After itemsets are investigated by Agrawal, Imieliński, and Swami[AIS93], and Agrawal and Srikant[AS94], the problem is a long-standing fundamental study to analyze theoretical and practical questions on pattern mining. Zaki[ZPOL97] investigated vertical representations of databases for efficient algorithms. Association rules based on frequent itemsets are a good example in the data mining community. Agrawal et al.[AMS+96] studied an efficient algorithm, and a generalization of association rules is also an important research topic in the area; e.g., generalized association rules[SA95] and evaluations of association rules on multi-level viewpoint[HF95]. Contests are held several times to investigate experimental performance of several algorithms (e.g.,[GZ03,JGZ04]).

To reduce the output of patterns, several algorithms have been developed for itemsets first, and they are generalized for various patterns. Algorithms for mining frequent maximal itemsets are MaxMiner[Bay98], Mafia[BCG01], and GenMax[GZ05]. Algorithms for mining frequent closed itemsets are CLOSET[PHM00], CLOSET+[WHP03], LCM[UAUA03,UKA04], CHARM[ZH02] and DCI Closed[LOP06].

[†]http://www-ikn.ist.hokudai.ac.jp/~wasa/enumeration_complexity.html is a page of listing the listing complexity of several listing algorithms

Rather than maximal and closed itemsets, *non-derivable patterns* by Calders and Goethals[CG02]. Selecting Top-k patterns in a lossy manner is also famous; e.g., Han et al.[HWLT02], Wang et al.[WHLT05], Xin et al.[XCYH06]. Approximation of the output by Afrati et al.[AGM04].

Sequential, Tree, and Graph patterns Agrawal ans Srikant[AS95] also studied the algorithm AprioriAll for *sequential patterns*, which take into account the precedence-subsequence relations among elements in $\mathcal{B}$. They in[SA96] studied the efficient algorithm, named GSP, based on[AS95], together with our background knowledge. Zaki studied the algorithm named SPADE[Zak01], which takes into account the equivalent classes in sequential pattern mining. Pei et al.[PHMa+01] studied the DFS-based algorithm algorithm named PrefixSpan for the efficient sequential pattern mining to avoid the candidate generation part in the Generation Phase of apriori-based algorithms. Closed patterns are also an important technique for sequential patterns. For example Pan et al. developed Carpenter[PCT+03] and Yan et al.[YHA03] studied CloSpan based on the PrefixSpan algorithm.

For more structured data, Inokuchi et al.[IWM00] introduced the Apriori-like pattern mining instance for graphs. Miyahara et al.[MSU+01] and Zaki[ZH02] studied its very efficient version for some classes of trees, which are a special case of graphs (without cycles). ILP-based studies (e.g.,[DTK98]) were also an example of data mining from graphs, where graphs are not represented in the graph, which are related to relational data in pattern mining.

Constraints and Compressions Exploratory paradigms in the Generation phase are important topics to reduce the computational complexity (e.g., Ng et al.[NLHP98]). To introduce and develop more useful interestingness measures as queries to transaction databases, constraints represented by several forms / formulae are also studied: Constraints for itemsets[SVA97], those by regular expressions by Garofalakis et al.[GRS99]. For Pattern Set Mining, several models have been studied: the maximum entropy model[JS02,Tat08], tiling[GGM04,GMS04,TV12], constraint-based methods[DRZ07], the minimum description length principle-based approaches[SVVL06,VvLS11].

Discriminative Pattern Mining

The above literatures are only about *unsupervised* pattern mining. For the supervised pattern mining problems, a.k.a. *discriminative pattern mining*[MS00,CYHH07]. Let us consider a (statistical) measure f and its parameter ι (e.g., the p-value in statistics) of getting information by the query to two positive and negative databases $\mathcal{D}^+$ and $\mathcal{D}^-$. Then the question is getting a theory in which all pattern satisfy both constraints; that from pattern mining and that from statistical measure, as desribed below.

$$Th(\mathcal{L}, \mathcal{D}^+, \mathcal{D}^-, q_{D^+}, q_{D^-}, f, \iota) = \{\phi \in \mathcal{L} \mid \begin{smallmatrix} q_D(\phi) \text{ is true, and} \\ f(q_{D^+}(\phi), q_{D^-}(\phi), D^+, D^-) \geq \iota \end{smallmatrix}\}.$$

The problem is also known (or, studied as different forms of) *mining emerging patterns*[DL99], *subgroup discovery*[Wro97,LKFT04], and *interesting pattern mining*[BAG00].

CHAPTER 3

Mining from Graphs of Bounded Treewidth

THE TARGET PATTERN MINING INSTANCE in this chapter is the *Frequent Connected Induced Subgraph Mining* (FCISM) problem, where the database is a set of graphs, a pattern language $\mathcal{L}$ is also a set of graphs, and *induced subgraph isomorphism*, an instance of graph matching operators, is used to evaluate patterns.

Our first question is how difficult the mining problem is on the viewpoint of the listing complexity. We then show that the FCISM problem cannot be solved for *arbitrary graphs* in output polynomial time if P $\neq$ NP. This result is followed by the second question: What constraints on graphs are helpful to achieve the efficient pattern mining algorithms. We in this chapter focus on the parameterization of graphs named *treewidth*, and deal with graphs of bounded treewidth (i.e., graphs whose treewidth parameters are less than or equal to a some constant k), we prove that the FCISM problem can be solved in *incremental polynomial time*.

keywords: Frequent Graph Mining, Graphs of Bounded Treewidth, Tree Decompositions, Dynamic Programming on Tree Decompositions

3.1. Introduction

Over the past 15 years a lot of research efforts have been devoted to build efficient and effective frequent graph mining algorithms (recall Chapter 2). Despite the studies in the literature, theoretical aspects of the topic have been still understudied. The importance of a better understanding of the theoretical complexity aspects of various graph mining problems appears somewhat neglected, which has negative side effects that most algorithms are limited to some ten thousands graphs only in transaction databases.

The goal of this chapter is to take one step towards a better understanding of graph pattern mining. Particularly, the *Frequent Connected Induced Subgraph Mining* (FCISM) problem is discussed. This is a mining instance of enumerating all connected graphs that are induced subgraph isomorphic to at least θ graphs in the

FCISM Problem

This chapter is based on the publication below:

- Horváth, T., Otaki, K., Ramon, J. (alphabetical order): Efficient Frequent Connected Induced Subgraph Mining in Graphs of Bounded Tree-Width, In Proc. of ECML/PKDD 2013 (LNCS 8188), pp.622-637, 2013, DOI:10.1007/978-3-642-40988-2_40.

given database $\mathcal{D}$, where all graphs in the output $\mathcal{F}$ are non-isomorphic in a pairwise manner. This problem, as we show, cannot be solved in output polynomial time for arbitrary transaction graphs. For forests, however, it can be solved in incremental polynomial time.

Graph of Bounded Treewidth

As the main result, we generalize the positive result on forests by showing that the FCISM problem can be solved in incremental polynomial time for graphs of *bounded treewidth*. The positive result of this paper is of practical, theoretical, and algorithmic importance. Regarding the practical aspects, we mention e.g. the ZINC dataset containing about 16.5 millions molecular graphs: 99.99% of these graphs have treewidth at most 3. With respect to its theoretical aspects, we note that induced subgraph isomorphism is a *persistent* problem that remain NP-complete even for graphs of treewidth 2[MT92]. Our positive result provides an example of the case that *efficient pattern mining is possible even for computationally intractable graph pattern matching operators*. To the best of our knowledge, there is only one further such example[HR10]. Finally, relating to the algorithmic aspects, the paradigm we followed, and which is used also in[HR10] for the frequent connected subgraph mining (FCSM) problem in graphs of bounded treewidth, appears sufficiently general for the design of graph mining algorithms for further graph pattern matching operators. This paradigm consists of the following main steps:

FCSM Problem

1. Give a *generic* levelwise search algorithm with some conditions,
2. prove the existence of an *efficiently* computable pattern refinement operator that is *complete* with respect to the graph pattern matching operator, and
3. show that the otherwise exponential-time dynamic-programming algorithm provided in[MT92] deciding the underlying graph pattern matching works in time *polynomial* in the size of the set of patterns generated so far.

When comparing the steps above with those in subgraph isomorphism[HR10], on the one hand one can notice a number of steps that are (almost) the same for the two problems. On the other hand, however, there are some crucial steps that require entirely different techniques, induced from the difference of graph matching operators. Thus, for example, the pattern refinement operator and the combinatorial characterization of the necessary information needed to calculate by the graph pattern matching algorithm become much more complicated for induced subgraph isomorphism.

3.2. Graphs and Tree Decompositions

Undirected Graph
Simple Graph
Labeled Undirected Graph

Graphs: An *undirected graph* is a pair (V, E), where V is a set of *vertices* and $E \subseteq \{\{u, v\} \mid u, v \in V\}$ is a set of (undirected) *edges*. We consider simple graphs that do not contain loops or parallel edges. A *labeled undirected graph* is a triple (V, E, λ), where (V, E) is an undirected graph and λ is the labeling function $\lambda : V \cup E \to \mathbb{N}$. The set of vertices, the set of edges, and the labeling function of a graph G are denoted by

$V(G)$, $E(G)$, and λ_G, respectively. A *subgraph* of G is a graph G' with $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$, and $\lambda_{G'}(x) = \lambda_G(x)$ for all $x \in V(G') \cup E(G')$; G' is an *induced subgraph* of G if it is a subgraph of G satisfying $\{u, v\} \in E(G')$ if and only if $\{u, v\} \in E(G)$ for all $u, v \in V(G')$. For $S \subseteq V(G)$, $G[S]$ denotes the induced subgraph of G with vertex set S. For $v \in V(G)$, $G \ominus v$ denotes $G[V(G) \setminus \{v\}]$. Unless otherwise stated, by graphs we mean *labeled undirected* graphs.

Subgraph

Induced Subgraph

A *path* connecting two vertices v_1, v_k of a graph G, denoted by P_{v_1,v_k}, is a sequence $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\} \in E(G)$ such that the v_i's are pairwise distinct. A graph is *connected* if there is a path between any pair of its vertices. A *connected component* of G is a maximal subgraph of G that is connected. The set of all connected components of G is denoted by $\mathcal{C}(G)$.

Path

Connected Graph

Connected Component

Graph morphisms: Two graphs G_1 and G_2 are *isomorphic*, denoted by $G_1 \simeq G_2$, if there is a *bijection* $\varphi : V(G_1) \to V(G_2)$ satisfying

Graph Isomorphism

1. $\{u, v\} \in E(G_1)$ if and only if $\{\varphi(u), \varphi(v)\} \in E(G_2)$ for every $u, v \in V(G_1)$,
2. $\lambda_{G_1}(u) = \lambda_{G_2}(\varphi(u))$ for every $u \in V(G_1)$, and
3. $\lambda_{G_1}(\{u, v\}) = \lambda_{G_2}(\{\varphi(u), \varphi(v)\})$ for every $\{u, v\} \in E(G_1)$.

For G_1 and G_2 we say that G_1 is *subgraph isomorphic* to G_2, denoted by $G_1 \preccurlyeq G_2$, if G_1 is isomorphic to a subgraph of G_2.

Subgraph Isomorphism

It is *induced subgraph isomorphic* to G_2, denoted by $G_1 \preccurlyeq_i G_2$, if it is isomorphic to an induced subgraph of G_2. In what follows, two graphs are regarded the same graph if they are isomorphic. By this mean, the FCISM problem should be solved without containing the same graphs in the output.

Induced Subgraph Isomorphic

Treewidth and Tree Decompositions: A central notion is *treewidth* that was reintroduced in algorithmic graph theory in [RS86]. A *tree decomposition* of a graph G, denoted by $TD(G)$, is a pair $TD(G) = (T, \mathcal{X})$, where T is a rooted tree and $\mathcal{X} = (X_z)_{z \in V(T)}$ is a family of subsets of $V(G)$ satisfying the conditions:

Treewidth

Tree Decomposition

1. $\cup_{z \in V(T)} X_z = V(G)$,
2. for every $\{u, v\} \in E(G)$, there is a $z \in V(T)$ such that $u, v \in X_z$, and

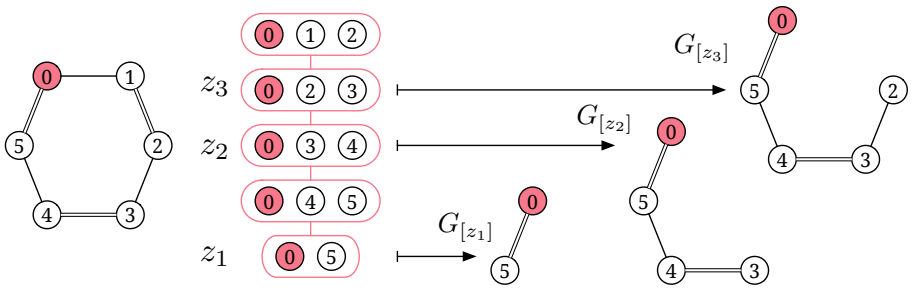


Figure 3.1.: An example of a tree decomposition with a cycle. The colored vertex 0 shows the example of the condition 3 of tree decompositions. A cycle has the treewidth 2. The right parts show examples of terminal graphs $G_{[z]}$.

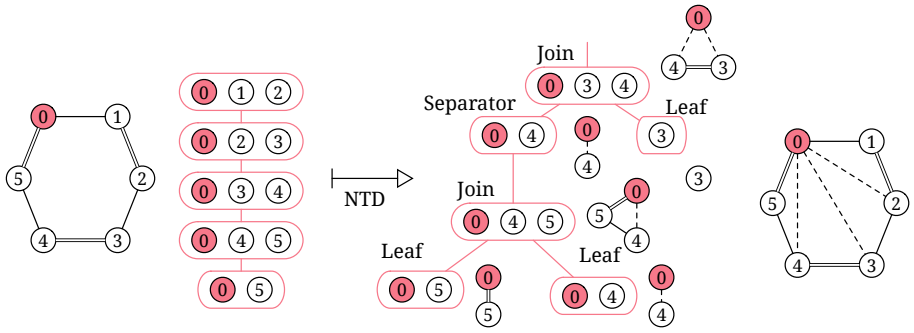


Figure 3.2.: A *TD*(G) in Figure 3.1 is transformed into a nice tree decomposition *NTD*(G).

3. $X_{z_1} \cap X_{z_3} \subseteq X_{z_2}$ for every $z_1, z_2, z_3 \in V(T)$ such that z_2 is on the path connecting z_1 with z_3 in T.

The set X_z associated with a node z of T is called the *bag* of z. The nodes of T will often be referred to as the nodes of the tree decomposition *TD*(G). The treewidth of $(T, \mathcal{X})$ is $\max_{z \in V(T)} |X_z| - 1$, and the *treewidth* of G, denoted by $\mathrm{tw}(G)$, is the minimum treewidth over all tree decompositions of G. For our discussions, $G_{[z]}$ denotes the induced subgraph of G defined by the union of the bags of z's descendants, where z is considered also as a descendant of itself. By graphs of bounded treewidth we mean graphs of treewidth at most k, where k is some constant. See an example in Figure 3.1 of a cycle of 6 vertices and a tree decomposition of $\mathcal{X} = \{\{0,1,2\},\{0,2,3\},\{0,3,4\},\{0,4,5\},\{0,5\}\}$.

Graph of Bounded Treewidth

We will use a special kind of tree decomposition, named a *nice tree decomposition* of G, denoted by *NTD*(G). It is a tree decomposition $(T, \mathcal{X})$, where T is a rooted binary tree consisting of only three types of nodes.

Nice Tree Decomposition

1. *leaf* node has no children,
2. *separator* node z has a single child z' with $X_z \subseteq X_{z'}$, and
3. *join* node z has two children z_1 and z_2 with $X_z = X_{z_1} \cup X_{z_2}$.

It follows from[Bod98] that for graphs of treewidth at most k, where k is some constant, a nice tree decomposition of treewidth at most k always exists and can be constructed in linear time. See an example of a nice tree decomposition in Figure 3.2. Intuitively, leaf nodes provide smallest fragments (i.e., subgraphs) in G, and join nodes merge two fragments and make a larger fragment. In addition, separator nodes can be recognized nodes to separate a graph into two parts; one part is vertices already processed (i.e., the vertex 5) and another part represents those processed later (i.e., the vertices 0 and 4). These interpretations are given by an idea of traversing *NTD*(G) from leaves to the root (i.e., in a postorder manner) in order to evaluate a graph matching operator (i.e., induced subgraph isomorphism), which will be described later sections.

Treewidth is a useful parameter of graphs, as many NP-hard problems can be solved in polynomial time particularly for graphs of bounded treewidth. However, well-known NP-complete problems – subgraph isomorphism and induced

subgraph isomorphism (e.g., [GJ79]) – remain NP-complete even for the easiest case in our setting; graphs of treewidth 2 [MT92,Sys82].

3.3. Frequent Connected Induced Subgraph Mining

We start by defining the pattern mining problem we are interested in.

FCISM Problem

Problem 3.1 (The FCISM problem) The *Frequent Connected Induced Subgraph Mining* (FCISM) problem is a pattern mining instance aiming at *listing* the set $\mathcal{F}$ of all *distinct* frequent connected *induced subgraphs*, that is, all connected graphs that are induced subgraph isomorphic to at least θ graphs in $\mathcal{D}$ when a class $\mathcal{G}$ of graphs, a transaction database $\mathcal{D}$ of graphs from $\mathcal{G}$, and an integer support count threshold $\theta > 0$ are *given*.

Note that we do not distinguish between isomorphic graphs. The *parameter* of the above problem is the size of $\mathcal{D}$. Clearly, the size of $\mathcal{F}$ can be exponential in that of $\mathcal{D}$*. Thus, in general, the set of all frequent connected induced subgraphs cannot be computed in time polynomial only in the size of $\mathcal{D}$. The following simple polynomial reduction shows that even output polynomial time enumeration is unlikely.

Theorem 3.1 (The Intractability on the FCISM problem) Unless P = NP, the FCISM problem cannot be solved in output polynomial time.

Proof We consider a reduction from the NP-complete k-Clique problem. For an unlabeled graph G with n vertices, let $\mathcal{D}$ consist of G and the clique K_n with n vertices. For $\mathcal{D}$ and $t = 2$, the number of frequent connected induced subgraphs is at most n (i.e., all cliques up to size n). Thus, if the FCISM problem could be solved in output polynomial time, we could decide the k-Clique problem in polynomial time by listing first the set $\mathcal{F}$ of all 2-frequent connected induced subgraphs and checking then whether $|\mathcal{F}| \geq k$ or not. ■

3.3.1. A Generic Levelwise Search Mining Algorithm

If, however, the transaction graphs are restricted to forests, the FCISM problem becomes equivalent to the FCSM problem, which can be solved with polynomial delay, and thus, in incremental polynomial time (see, e.g., [HR10]). Our goal in this paper is to show that the FCISM problem can be solved in incremental polynomial time for graphs of *bounded treewidth*. To prove this result, we start by giving a generic algorithm, called FCISM, that lists frequent connected induced subgraphs with levelwise search (see Algorithm 5, the special case of the Levelwise algorithm

*Take for instance the database consisting of a single labeled clique over n vertices such that each vertex has a unique label and set the frequency threshold θ to 1, inducing the polynomial number of subgraphs.

Algorithm 5 The FCISM ALGORITHM

Input: A transaction database $\mathcal{D}$ and a threshold $\theta > 0$.

Call FCISM ($\mathcal{D}, \mathcal{L}, \preccurlyeq, q_D$).

1: **procedure** FCISM ($\mathcal{D}$, θ)
2: let $\mathcal{F}_1 \subseteq \mathcal{G}$ be the set of frequent graphs consisting of a single labeled vertex
3: **for** ($l := 1$; $\mathcal{F}_l \neq \emptyset$; $l := l + 1$) **do**
4: $C_{l+1} := \mathcal{F}_{l+1} := \emptyset$
5: **for all** $P \in \mathcal{F}_l$ **do**
6: **for all** $H \in \rho(P) \cap \mathcal{G}$ satisfying (i) $H \notin C_{l+1}$ and (ii) $\rho^{-1}(H) \subseteq \mathcal{F}_l$ **do**
7: add H to C_{l+1}
8: **print** H and add it to $\mathcal{F}_{l+1}$ **if** $|\{G \in \mathcal{D} : H \preccurlyeq_i G\}| \geq t$

in Algorithm 4 described with the refinement operator ρ). As we will discuss below in this section, the difference of the refinement operator ρ arises a completely different complexity issue for the FCISM problem that is not raised by the FCSM problem. The algorithm assumes the transaction graphs to be elements of some graph class $\mathcal{G}$ that is closed under taking subgraphs. Thus, as we are interested in mining frequent connected induced subgraphs, all patterns belong to $\mathcal{G}$ as well.

Levelwise Search Algorithm

Recall that one of the basic features of the levelwise search algorithms is that the underlying pattern language $\mathcal{L}$ is associated with *naturally defined partial order*.

Generalization

Following the common pattern mining terminology (see, e.g.,[MT97], Section 2.3), for a pattern language $(\mathcal{L}, \leq)$ we say that a pattern $P_1 \in \mathcal{L}$ is a *generalization* of a pattern $P_2 \in \mathcal{L}$ (or P_2 is a *specialization* of P_1) if $P_1 \leq P_2$; P_1 is a *proper generalization*

Proper Generalization

of P_2 (or P_2 is a *proper specialization* of P_1), denoted by $P_1 < P_2$, if $P_1 \leq P_2$ and $P_1 \neq P_2$. Furthermore P_1 is a *direct generalization* of P_2 (or P_2 is a *direct specialization* of P_1) if $P_1 < P_2$ and there is no $P_3 \in \mathcal{L}$ with $P_1 < P_3 < P_2$. Now the underlying pattern language $\mathcal{L}$ is the set of all finite connected (labeled) graphs of $\mathcal{G}$, associated with the following natural generalization relation $\leq$ defined as follows: For any $P_1, P_2 \in \mathcal{L}$, $P_1 \leq P_2$ if and only if $P_1 \preccurlyeq_i P_2$. The proofs of the two claims in the proposition below are straightforward.

Proposition 3.1 Let $\mathcal{L}$ and $\leq$ be as defined above. Then $(\mathcal{L}, \leq)$ is a partially ordered set. Furthermore, for any $P_1, P_2 \in \mathcal{L}$ it holds that P_1 is a direct generalization of P_2 if and only if

$$P_1 < P_2 \text{ and } |V(P_1)| = |V(P_2)| - 1. \tag{3.1}$$

In the main loop of Algorithm 5 (lines 4–8), the set $\mathcal{F}_{l+1}$ of frequent connected induced subgraphs containing $l + 1$ *vertices* are calculated from those containing l *vertices*, in accordance with condition (3.1). In particular, for each frequent pattern $P \in \mathcal{F}_l$, we first compute a set $\rho(P) \cap \mathcal{G}$ of graphs, where $\rho(P)$ is a subset of the set of *direct* specializations of P. Clearly, the graphs in $\rho(P)$ are all connected by the choice of $\mathcal{L}$. Notice that we cannot define $\rho(P)$ as the set of *all* direct specializations of P, as this set can be of exponential cardinality. In Theorem 3.2 below we will provide sufficient conditions for ρ needed for efficient pattern enumeration.

For each direct specialization $H \in \rho(P) \cap \mathcal{G}$, we check whether it has already been generated during the current iteration (see condition (i) in line 5). If not, we also check for each connected direct generalization of H, denoted by $\rho^{-1}(H)$ in the algorithm, whether it is frequent (condition (ii) in line 5). Here we utilize that frequency is an anti-monotonic interestingness predicate for $(\mathcal{L}, \leq)$. In what follows, candidate patterns generated by Algorithm 5 that satisfy conditions (i) and (ii) in line 5 will be referred to as *strong candidates*. If H is a strong candidate, we add it to the set C_{l+1} of candidate patterns consisting of $l+1$ *vertices* and compute its support count (lines 6–7). If H is frequent, i.e., it is induced subgraph isomorphic to at least t transaction graphs in $\mathcal{D}$, we add it to the set $\mathcal{F}_{l+1}$.

Strong Candidate

By Theorem 3.1 above, the FCISM problem cannot be solved in output polynomial time for the general problem setting. If, however, the class $\mathcal{G}$ of transaction graphs and the *refinement operator* ρ satisfy the conditions of Theorem 3.2 below, the FCISM problem can be solved in incremental polynomial time. To state the theorem, we recall some basic notions for refinement operators (see, e.g., [NCW97]). A downward refinement operator Ξ for a poset $(\mathcal{L}, \leq)$ is a function $\Xi : \mathcal{L} \to 2^{\mathcal{L}}$ with $\Xi(P) \subseteq \{P' : P \leq P'\}$ for all $P \in \mathcal{L}$. That is, $\Xi(P)$ is a subset of the set of specializations of P [†] For Ξ, we define the n-th power $\Xi^n : \mathcal{L} \to 2^{\mathcal{L}}$ recursively by

$$\Xi^n(P) = \begin{cases} \Xi(P) & \text{if } n = 1 \\ \Xi(\Xi^{n-1}(P)) & \text{otherwise} \end{cases}$$

for all $n \in \mathbb{N}$. Finally, we say that Ξ is *complete*, if for all $P \in \mathcal{L}$, there is some $n \in \mathbb{N}$ with $P \in \Xi^n(\bot)$, where $\bot$ denotes the empty graph.

Using the above notions, we can formulate the following generic theorem:

Theorem 3.2 (Generic Levelwise Search) Let $\mathcal{G}$ be a class of graphs, $\mathcal{L}$ be the set of connected graphs in $\mathcal{G}$, and $\rho : \mathcal{L} \to 2^{\mathcal{L}}$ be a downward refinement operator. If ρ and $\mathcal{G}$ satisfy the conditions below then Algorithm 5 solves the FCISM problem in incremental polynomial time and in incremental polynomial space.

(i) $\mathcal{G}$ is closed under taking subgraphs.
(ii) The membership problem in $\mathcal{G}$ can be decided in polynomial time.
(iii) ρ is complete and $\rho(P)$ can be computed in time polynomial in the combined size of the input and the set of frequent patterns listed so far by Algorithm 5.
(iv) Isomorphism can be decided in polynomial time for $\mathcal{G}$.
(v) For every $H, G \in \mathcal{G}$ such that H is connected, it can be decided in time polynomial in the combined size of the input and the set of frequent patterns listed so far by Algorithm 5 whether $H \preccurlyeq_i G$.

[†] We note that refinement operators based on pattern specializations are referred to as *downward* refinement operators in Inductive Logic Programming (ILP).

Treewidth	number of molecules	percentage (%)
1	38,754	0.23
2	16,349,503	99.08
3	111,764	0.68
≥ 4	1,313	0.01

Table 3.1.: Treewidth distribution of molecular graphs in the ZINC dataset.

Proof The proof follows directly from the remarks and concepts above. ■

The following positive result on forests can immediately be obtained by applying the theorem above (it follows also e.g. from [HR10]):

Corollary 3.1 The FCISM problem can be solved in incremental polynomial time for forest transaction graphs.

Proof Conditions (i)–(iii) of Theorem 3.2 are straightforward for forests and conditions (iv)–(v) holds by the fact that induced subgraph isomorphism from a tree into a forest can be decided in polynomial time (see, e.g., [Mat78]).

3.4. Mining Graphs of Bounded Treewidth

We generalize the positive result of Corollary 3.1 to graphs of bounded treewidth and prove the main result:

Theorem 3.3 The FCISM problem can be solved in incremental polynomial time for graphs of bounded treewidth.

Before proving this result, we note that the class of graphs of bounded treewidth is not only of theoretic interest, but also of practical relevance as we explained. As an example, consider the ZINC dataset[‡] consisting of more than 16 million chemical compounds. Regarding the distribution of the molecular graphs with respect to their treewidth, 99.99% of the $16,501,334$ molecular graphs in this dataset have treewidth at most 3 and $99,31$% only treewidth at most 2. See Table 3.1.

To prove Theorem 3.3, the positive mining result on graphs of bounded treewidth, it suffices to show that all conditions of Theorem 3.2 hold for graphs of bounded treewidth. The proof of the claims in the theorem below is shown in [HR10] for the positive result in graphs of bounded treewidth; notice that the conditions considered in the theorem are all independent of the underlying graph pattern matching operator.

[‡]We used a commercial version of the ZINC dataset for the treewidth statistics.

Theorem 3.4 Conditions (i), (ii), and (iv) of Theorem 3.2 hold for the class of graphs of bounded treewidth.

Proof (i) It is easy to see that the class of graphs of treewidth at most k is closed under taking subgraphs.
(ii) For any constant k, it can be decided in linear time whether a graph has treewidth at most k [Bod96].
(iv) Isomorphism can be decided in polynomial time for graphs of bounded treewidth [Bod90].

■

Thus, to show that the FCSM problem can be solved in incremental polynomial time for graphs of bounded treewidth, we only need to prove conditions (iii) and (v). We first show (iii).

Theorem 3.5 For the class of graphs of bounded treewidth, there exists a refinement operator ρ satisfying condition (iii) of Theorem 3.2.

Proof For a connected pattern P with $\mathrm{tw}(P) \leq k$, define the refinement $\rho(P)$ of P as follows: A connected graph P' with $\mathrm{tw}(P') \leq k$ is in $\rho(P)$ iff P' has a vertex v with degree at most k such that $P \simeq P' \ominus v$. Notice that this definition is unique, as isomorphic graphs are not distinguished from each other by definition. Clearly, $\rho(P)$ is a subset of the set of direct specializations of P. Utilizing condition (i) of Theorem 3.2 and the basic fact that every graph of treewidth at most k has a vertex of degree at most k, the completeness of ρ follows directly by induction on the number of vertices.

We now turn to the complexity of computing $\rho(P)$ and show the stronger property that $\rho(P)$ can actually be computed in time polynomial in the size of $\mathcal{D}$. Since each new vertex v is connected to P by at least one and at most k vertices, for the cardinality of $\rho(P)$ we have

$$|\rho(P)| \leq \sum_{i=1}^{k} |\Lambda|^{i+1} \binom{n}{i} < |\Lambda|^{k+1}(n+1)^k,$$

where Λ is the set of vertex and edge labels used in $\mathcal{D}$ and n is the number of vertices of P. Since k is a constant, $|\rho(P)|$ is polynomial in the size of $\mathcal{D}$, and hence, as condition (iv) of Theorem 3.2 holds by Theorem 3.4, $\rho(P)$ can be computed in time polynomial in the size of $\mathcal{D}$, as claimed. ■

It remains to show for the proof of Theorem 3.3 that condition (v) also holds. In Section 3.4.1 we first recall from [HN07] a dynamic programming algorithm deciding induced subgraph isomorphism for a restricted class of graphs of bounded treewidth. Given a connected graph H and a transaction graph G, both of bounded treewidth, this algorithm decides $H \preccurlyeq_i G$ by computing recursively a certain set of tuples representing *partial* induced subgraph isomorphisms between H and G. The

problem is, however, that for arbitrary graphs of bounded treewidth, the number of such partial solutions can be exponential in the size of H. Using the paradigm developed in[HR10] for frequent connected subgraph mining in graphs of bounded treewidth, we will show that $H \preccurlyeq_i G$ can be decided by computing only a polynomial number of *new* partial solutions and efficiently recovering all missing partials solutions from those calculated for the already generated frequent patterns.

3.4.1. A Dynamic Programming Algorithm

The dynamic programming algorithm from[HN07] decides induced subgraph isomorphism for a restricted class of graphs of bounded treewidth. It is based on an efficient algorithm[MT92] deciding various morphisms between bounded treewidth *and bounded degree graphs*, which, in turn, follows a generic dynamic programming approach designed in[AP89]. In order to be consistent with[HR10] on frequent connected subgraph mining in graphs of bounded treewidth, we naturally adapt the notions and notations from Section 4.1 of[HR10].

Let H and G denote connected graphs with $\mathrm{tw}(H), \mathrm{tw}(G) \leq k$. The algorithm in[HN07] decides whether $H \preccurlyeq_i G$ by computing a nice tree decomposition *NTD*(G) of G, traversing *NTD*(G) in a postorder manner, calculating for each node in *NTD*(G) a set of tuples, called *characteristics*, and by testing whether the root of *NTD*(G) has a characteristic satisfying a certain condition formulated in Lemma 3.1 below. In the dynamic programming, partial answers of partial problems (i.e., problems of deciding the matching for small subgraphs) are stored in the form of *iso-quadruples*.

Iso-Quadruple

More precisely, an *iso-quadruple* of H relative to a node z of *NTD*(G) is a quadruple $(S, \mathcal{R}, K, \psi)$, where

(i) $S \subseteq V(H)$ with $|S| \leq k+1$,
(ii) $\mathcal{R} \subseteq \mathcal{C}(H[V(H) \setminus S])$,
(iii) $K = H[S \cup V(\mathcal{R})]$, and
(iv) $\psi : S \to X_z$ is an *induced* subgraph isomorphism from $H[S]$ to $G[X_z]$.

The set of all iso-quadruples of H relative to a node z of *NTD*(G) is denoted by $\Gamma(H, z)$. See also Figure 3.3 for example. For a node z in *NTD*(G), an iso-quadruple $(S, \mathcal{R}, K, \psi) \in \Gamma(H, z)$, if there exists an induced subgraph isomorphism φ from K to $G_{[z]}$ satisfying the following condition, the iso-quadruple is z-*characteristic*:

Characteristic

(i) $\varphi(u) = \psi(u)$ for all $u \in S$ and
(ii) $\varphi(v) \notin X_z$ for all $v \in V(\mathcal{R})$.

These conditions imply that $\varphi(u) \in X_z$ for all $u \in S$. In addition, the set of all z-characteristics of H relative to z is denoted by $\Gamma_{\mathrm{ch}}(H, z)$. Clearly, $\Gamma_{\mathrm{ch}}(H, z) \subseteq \Gamma(H, z)$. The following lemma from[HN07] provides a characterization of induced subgraph isomorphism in terms of r-characteristics for the root r of *NTD*(G).

Lemma 3.1 Let r be the root of a nice tree decomposition *NTD*(G) of G. Then $H \preccurlyeq_i G$ iff there exists $(S, \mathcal{R}, K, \psi) \in \Gamma_{\mathrm{ch}}(H, r)$ with $K = H$.

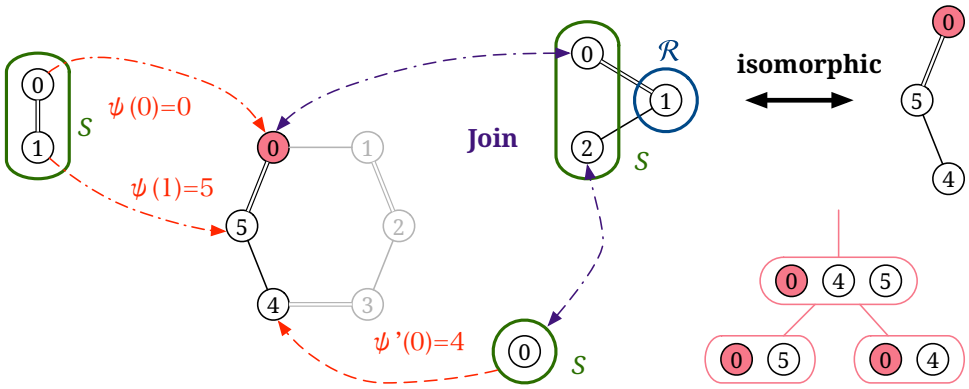


Figure 3.3.: An iso-quadruple of the graph of 3 vertices and 2 edges, where $S = \{0, 1\}$. An example of the mapping from S to $X_z = \{0, 5\}$ is $\psi(0) = 0$ and $\psi(1) = 5$. Two iso-quadruples on $X_z = \{0, 5\}$ and $X_{z'} = \{0, 4\}$ are merged to get a larger iso-quadruple in $X_{z''} = \{0, 4, 5\}$.

Thus, by the lemma above, we need to calculate the characteristics of the root of $NTD(G)$. Lemma 3.2 below from[HN07] shows how to compute the set of characteristics for leafs. In addition, it explains how to compute $\Gamma_{cf}(H, z)$ for a separator or a join node z from characteristics in its children. This enables the computation of the characteristics for all nodes of $NTD(G)$ by a postorder traversal of $NTD(G)$. Figure 3.3 illustrates basic ideas for the join node $\{0, 4, 5\}$ of $NTD(G)$.

Lemma 3.2 Let G, H be connected graphs of bounded treewidth and z be a node in $NTD(G)$. For all $(S, \mathcal{R}, K, \psi) \in \Gamma(H, z)$ it holds that $(S, \mathcal{R}, K, \psi) \in \Gamma_{ch}(H, z)$ if and only if one of the following conditions holds:

LEAF: z has no children and $\mathcal{R} = \emptyset$.

SEPARATOR: z has a single child z' and $\exists (S', \mathcal{R}', K', \psi') \in \Gamma_{ch}(H, z')$ with

(S.a) $S = \{v \in S' : \psi'(v) \in X_z\}$,
(S.b) $\mathcal{R}' = \{D' \in \mathcal{C}(H[V(H) \setminus S']) : D'$ is a subgraph of some $D \in \mathcal{R}\}$,
(S.c) $\psi(v) = \psi'(v)$ for every $v \in S$.

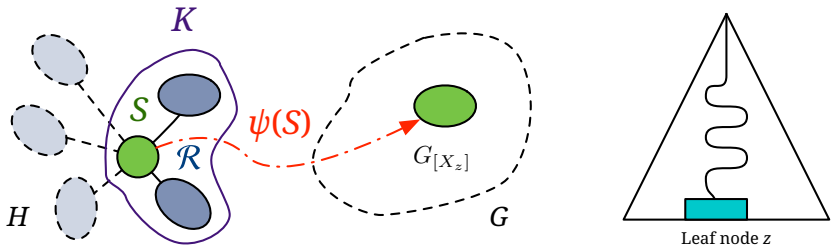
JOIN: z has two children z_1, z_2 and there exist $(S_1, \mathcal{R}_1, K_1, \psi_1) \in \Gamma_{ch}(H, z_1)$ and $(S_2, \mathcal{R}_2, K_2, \psi_2) \in \Gamma_{ch}(H, z_2)$ satisfying

(J.a) $S_i = \{v \in S : \psi(v) \in X_{z_i}\}$ for $i = 1, 2$,
(J.b) $\mathcal{R}_1$ and $\mathcal{R}_2$ form a binary partition of the connected components of $\mathcal{R}$,
(J.c) $\psi_i(v) = \psi(v)$ for every $v \in S_i$ and for $i = 1, 2$.

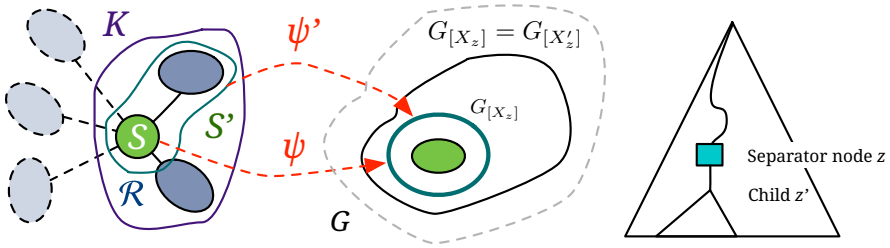
As mentioned, Lemma 3.2 provides a polynomial time algorithm to decide induced subgraph isomorphism for restricted subclasses of graphs of bounded treewidth (e.g., when the degree is bounded[MT92], or when the graphs have log-bounded fragmentation[HN07]). Clearly, the algorithm above is exponential for arbitrary graphs of bounded treewidth; this follows directly from the negative result in[MT92].

It is important to stress that almost the same notions and conditions are used for the Frequent *Connected Subgraph* Mining (FCSM) problem (cf. Section 4.1 in[HR10]). The only difference is in the definition of iso-quadruples, in particular, in the def-

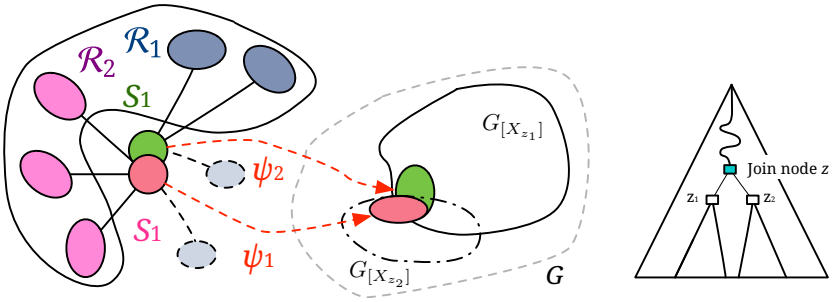
FCSM Problem



(a) Leaf node; As $|X_z| \leq k+1$, the all possible characteristics can be enumerated as $G_{[X_z]} = G[X_z]$, $\mathcal{R}$ should be $\emptyset$, and ψ is a isomorphism from $H[S]$ to $G[X_z]$.



(b) Separator node; It holds that $G_{[X_z]} = G_{[X'_z]}$ and it holds that $\psi = \psi'$ on the set S.



(c) Join node; merging two partial solutions with keeping the mapping (i.e., characteristics) on the common part $S = S_1 \cap S_2$.

Figure 3.4.: Illustrations on the dynamic programming approach for three types of nodes in the nice tree-decomposition.

inition of ψ, in accordance with the semantic difference between the FCSM and FCISM problems. However, as it turns out in Section 3.4.2 below, we need a different combinatorial arguments to show the positive result for the FCISM problem.

3.4.2. Feasible Iso-Quadruples

The source of computational intractability of the algorithm based on Lemma 3.2 is the possibly exponential number of iso-quadruples needed to test. Using the paradigm developed for the FCSM problem[HR10], we show that for each node of *NTD*(G), it suffices to check only a polynomial number of iso-quadruples, as we can utilize the characteristics of the frequent patterns computed earlier by Algorithm 5. Note that for all transaction graphs we fix a nice tree decomposition computed in a pre-processing part of the whole KDD process.

Let H_1, H_2, and G be connected graphs of bounded treewidth, *NTD*(G) be some fixed nice tree decomposition of G, and z be a node in *NTD*(G). For any two $\xi_1 = (S_1, \mathcal{R}_1, K_1, \psi_1) \in \Gamma(H_1, z)$ and $\xi_2 = (S_2, \mathcal{R}_2, K_2, \psi_2) \in \Gamma(H_2, z)$, ξ_1 is *equivalent* to ξ_2,

denoted by $\xi_1 \equiv \xi_2$, if there is an isomorphism π between K_1 and K_2 such that π is a bijection between S_1 and S_2 and $\psi_1(v) = \psi_2(\pi(v))$ for every $v \in S_1$. The lemma below shows that it suffices to store only one representative z-characteristic for each equivalence class of the set of z-characteristics and that equivalence between iso-quadruples can be decided in polynomial time. The proof is similar to that of the corresponding lemma in [HR10].

Lemma 3.3 Let G, H_1, and H_2 be connected graphs of treewidth at most k, z be a node in *NTD*(G), and $\xi_i = (S_i, \mathcal{R}_i, K_i, \psi_i) \in \Gamma(H_i, z)$ $(i = 1, 2)$. Then

(i) $\xi_1 \in \Gamma_{\text{ch}}(H_1, z)$ iff $\xi_2 \in \Gamma_{\text{ch}}(H_2, z)$ whenever $\xi_1 \equiv \xi_2$ and
(ii) $\xi_1 \equiv \xi_2$ can be decided in time $O\left(n^{k+4.5}\right)$.

For a strong candidate pattern H generated by Algorithm 5, $\mathcal{F}^{(H)}$ denotes the set of patterns consisting of H and all frequent patterns listed before H. For a transaction graph G and node z of *NTD*(G), an iso-quadruple $\xi \in \Gamma(H, z)$ of a strong candidate H is *redundant* if there are $P \in \mathcal{F}^{(H)} \setminus \{H\}$ and $\xi' \in \Gamma(P, z)$ with $\xi \equiv \xi'$; otherwise, ξ is *non-redundant*. Finally, $\Gamma_{\text{nr}}(H, z)$ and $\Gamma_{\text{nr,ch}}(H, z)$ denote the set of non-redundant iso-quadruples of H relative to a node z in *NTD*(G) and the set of non-redundant z-characteristics of H, respectively.

Proposition 3.2 below implies that for a strong candidate H and an iso-quadruple $\xi \in \Gamma(H, z)$, it has to be tested whether ξ is a z-characteristic of *NTD*(G) only when ξ is non-redundant; otherwise, it suffices to check whether ξ is equivalent to a non-redundant z-characteristic for some frequent pattern $P \in \mathcal{F}^{(H)} \setminus \{H\}$.

Proposition 3.2 Let H be a strong candidate, G be a transaction graph, both of bounded treewidth, and let $\xi \in \Gamma(H, z)$ for some node z in *NTD*(G). Then $\xi \in \Gamma_{\text{ch}}(H, z)$ if and only if there exists $\xi' \in \bigcup_{P \in \mathcal{F}^{(H)}} \Gamma_{\text{nr,ch}}(P, z)$ with $\xi \equiv \xi'$.

Thus, induced subgraph isomorphism can be decided by using only the non-redundant z-characteristics of the frequent patterns. Instead of non-redundant iso-quadruples, we can use an efficiently computable superset of them, the set of *feasible* iso-quadruples.

Feasible Iso-Quadruple

Lemma 3.4 Let H, G, and z be as in Proposition 3.2 and let $\xi \in \Gamma_{\text{nr}}(H, z)$ with $\xi = (S, \mathcal{R}, K, \psi)$. Then, for all vertices $v \in V(H) \setminus V(K)$ it holds that (i) the degree of v in H is at least 2 and (ii) v is a cut vertex in H.

Proof The proof of (i) applies a similar argument used for ordinary subgraph isomorphism [HR10]. In particular, suppose for contradiction that $V(H) \setminus V(K)$ has a vertex v with degree 1 in H. Since, by assumption, H contains at least one edge and is connected, it has no isolated vertices. Let H' be the graph obtained from H by removing v and the (only) edge adjacent to it. Clearly, H' is a connected induced subgraph of H. Since H is a strong candidate pattern, H' is a frequent

connected induced subgraph and has therefore already been generated by Algorithm 5. Furthermore, K is an induced subgraph of H' implying $\xi \in \Gamma(H', z)$. But ξ is then redundant for H, contradicting the assumption.

To prove (ii), suppose there is a non-cut vertex $v \in V(H) \setminus V(K)$ of H. Let $H' = H \ominus v$. Since H is connected and v is a non-cut vertex of H, H' is connected. Similarly to the case above, it holds that H' contains K as an induced subgraph because all edges that have been removed are outside of $E(K)$. Thus, $\Gamma(H', z)$ has an element equivalent to ξ, contradicting that ξ is non-redundant. ■

We now show that for any $S \subseteq V(H)$ with $|S| \leq k+1$, only a constant number of connected components in $H[V(H) \setminus S]$ can fulfill the two conditions of Lemma 3.4. Although the statement formulated below is similar to the corresponding claim stated for the case of ordinary subgraph isomorphism in [HR10], the arguments used in the proofs are entirely different, due to the difference of isomorphisms.

Lemma 3.5 Let H be a strong candidate pattern generated by Algorithm 5, $S \subseteq V(H)$ with $|S| \leq k+1$, and $\mathcal{C}_A$ be the set of connected components C from $\mathcal{C}(H[V(H) \setminus S])$ such that for all $v \in C$, v satisfies both conditions of Lemma 3.4. Then

$$|\mathcal{C}_A| \leq \binom{k+1}{2}.$$

To show the claim above, we first prove two technical lemmas.

Lemma 3.6 Let H, S, and $\mathcal{C}_A$ be as defined in Lemma 3.5. Then for all $C \in \mathcal{C}_A$, C is connected to S by at least two edges ending in different vertices in S.

Proof The claim is straightforward if $|V(C)| = 1$; H has no parallel edges by construction and the only vertex of C for this case must be connected to S by at least two edges, as it is a cut vertex in H.

The proof of the case $|V(C)| > 1$ utilizes the fact that every connected graph has at least two non-cut vertices. More precisely, let u be a non-cut vertex of C. Since u is a cut vertex in H by condition (i) of Lemma 3.4, it must be the case that u is connected to at least one vertex in S. Thus, C is connected to S by at least two edges, as it has at least two non-cut vertices. Suppose that all non-cut vertices of C are adjacent to the same vertex in S, say w. Let $u, v \in V(C)$ be different non-cut vertices of C. Since, on the one hand, u is a non-cut vertex of C, and, on the other hand, it is a cut vertex in H by the condition of the lemma, there are two vertices $x, y \in V(H)$ that are disconnected by u (i.e., x and y belong to different connected components of $H \ominus u$). Since u is a non-cut vertex of C, at most one of x and y can belong to C. It can easily be seen for this case that in fact, *exactly* one of x and y, say x, belongs to C. Furthermore, $\{u, w\}$ must be an edge on the path connecting x and y in H, i.e., x and y are connected by a path of the form $P_{x,u} + \{u, w\} + P_{w,y}$, where $P_{x,u}$ is a path in C. Since $C \ominus u$ remains connected, there is a path $P_{x,v}$ connecting x and v in $C \ominus u$. Thus, the path $P_{x,v} + \{v, w\} + P_{w,y}$

connects x and y in $H \ominus u$, contradicting that u disconnects x and y. Hence, all connected components in $\mathcal{C}_A$ are connected to at least two different vertices in S, as stated. ■

The second lemma states that each connected component of $\mathcal{C}_A$ *connects* such two vertices of S that are not *connected* by any other component of $\mathcal{C}_A$.

Lemma 3.7 Let H, S, and $\mathcal{C}_A$ be as defined in Lemma 3.5. Then for all connected components $C \in \mathcal{C}_A$, there exist $u', v' \in S$ such that

(i) $u' \neq v'$ and $\{u, u'\}, \{v, v'\} \in E(H)$ for some $u, v \in V(C)$, and
(ii) for all $C' \in \mathcal{C}_A \setminus \{C\}$, at least one of u' and v' is not adjacent to C'.

Proof By Lemma 3.6, for all $C \in \mathcal{C}_A$ there are $u', v' \in S$ satisfying (i). Thus, to show the claim above, suppose for contradiction that there exists a connected component $C \in \mathcal{C}_A$ with the following property: for all $u', v' \in S$ satisfying (i) for C, there is a C' such that ***both*** u' and v' are adjacent to C'. Let u be the only vertex of C if $|V(C)| = 1$; otherwise let u be a non-cut vertex of C. Since $C \in \mathcal{C}_A$, u is a cut vertex in H by condition and hence, there are $x, y \in V(H)$ such that u disconnects x and y in H. Depending on the number of vertices of C and on the membership of x and y in C, we distinguish the following cases by noting that the case $x, y \in V(C)$ cannot occur by the choice of u:

Case 1. Suppose $|V(C)| = 1$. Then x and y must be connected in H by a path of the form $P_{x,v} + \{v, u\} + \{u, w\} + P_{w,y}$ for some $v, w \in S$ with $v \neq w$, where the length of $P_{x,v}$ and $P_{w,y}$ can be zero. By assumption, v and w are adjacent to some $C' \in \mathcal{C}_A$ and thus, there is a path $P_{v,w}$ in H that does not contain u. But x and y are then connected in H by the path $P_{x,v} + P_{v,w} + P_{w,y}$, contradicting that u disconnects x and y.

Case 2. Suppose $x \in V(C)$. Then x cannot be a non-cut vertex of C, as in this case x must be adjacent to a vertex $x' \in S$, which, in turn, is not adjacent to u. It can be shown in a way similar to the proof of Case 1, that for this case there is a path in H connecting x and y that does not contain u; a contradiction. Thus, as C has at least two non-cut vertices if $|V(C)| > 1$, there is a non-cut vertex $v \in V(C)$ with $v \neq u$. Since u disconnects x and y in H, there is a path of the form $P_{x,u} + \{u, u'\} + P_{u',y}$, where $P_{x,u}$ is a path in C, $\{u, u'\}$ is an edge of H with $u' \in S$, and $P_{u',y}$ is a path connecting u' and y in H. Let $v' \in S$ be a vertex adjacent to v. Notice that $v' \neq u'$, as otherwise the path $P_{x,v} + \{v, u'\} + P_{u',y}$ connects x and y in $H \ominus u$, contradicting that u disconnects x and y; clearly, a path $P_{x,v}$ connecting x and v in $C \ominus u$ always exists, as u is a non-cut vertex of C. Thus, u', v' fulfill condition (i) and hence, u' and v' are connected by a path $P_{u',v'}$ via some connected component $C' \in \mathcal{C}_A$ by assumption. But then x and y are connected by the path $P_{x,v} + \{v, v'\} + P_{v',u'} + P_{u',y}$ in $H \ominus u$, a contradiction.

Case 3. The case of $x, y \notin V(C)$ can be shown in a way similar to the cases above.

■

The proof of Lemma 3.5 follows directly from Lemma 3.7 and $|S| \leq k+1$. With *feasible* iso-quadruples, a superset of non-redundant iso-quadruples, we formulate in Theorem 3.6 the main result, which states that feasible iso-quadruples can be used correctly to decide induced subgraph isomorphism and that the number of feasible iso-quadruples of a strong candidate pattern is polynomial in the pattern's size: For a strong candidate H generated by Algorithm 5 and for a node z in *NTD*(G) of a transaction graph of bounded treewidth, an iso-quadruple $\xi \in \Gamma(H, z)$ is called *feasible* if it satisfies the conditions of Lemma 3.4. The set of feasible iso-quadruples relative to z and the set of feasible z-characteristics are denoted by $\Gamma_{\mathrm{f}}(H, z)$ and $\Gamma_{\mathrm{f,ch}}(H, z)$, respectively.

Theorem 3.6 Let H be a strong candidate pattern generated by Algorithm 5 and z be a node of *NTD*(G) for some transaction graph G with $\mathrm{tw}(G) \leq k$. Then

(i) $\Gamma_{\mathrm{nr}}(H, z) \subseteq \Gamma_{\mathrm{f}}(H, z)$,
(ii) for all $\xi \in \Gamma(H, z)$, $\xi \in \Gamma_{\mathrm{ch}}(H, z)$ iff there exist a $\xi' \in \bigcup_{P \in \mathcal{F}^{(H)}} \Gamma_{\mathrm{f,ch}}(P, z)$ with $\xi \equiv \xi'$,
(iii) $|\Gamma_{\mathrm{f}}(H, z)| = O\left(|V(H)|^{k+1}\right)$, and
(iv) $\Gamma_{\mathrm{f}}(H, z)$ can be computed in time polynomial in the size of H.

Proof The proof of (i) is immediate from the definitions and Lemma 3.4. Claim (ii) follows from Proposition 3.2 and from the fact that $\bigcup_{P \in \mathcal{F}^{(H)}} \Gamma_{\mathrm{nr,ch}}(P, z)$ and $\bigcup_{P \in \mathcal{F}^{(H)}} \Gamma_{\mathrm{f,ch}}(P, z)$ are equal up to equivalence. To show (iii), let $S \subseteq V(H)$ with $|S| \leq k+1$ and $\mathcal{C}_A$ be the set of connected components as defined in Lemma 3.5. By definition, for every $\xi = (S, \mathcal{R}, K, \psi) \in \Gamma_{\mathrm{f}}(H, z)$, $\mathcal{R}$ contains all connected components in $\mathcal{C}(H[V(H) \setminus S])$ that are not in $\mathcal{C}_A$. For a fixed subset $S \subseteq V(H)$ with $|S| \leq k+1$ and for a fixed injective function $\psi : S \to X_z$ the number of possible feasible quadruples is bounded by $2^{|\mathcal{C}_A|}$, which, in turn, is bounded by $2^{\binom{k+1}{2}}$ by Lemma 3.5. The number of induced subgraph isomorphisms from $H[S]$ to $G[X_z]$ is at most the number of injective functions from S to the bag X_z of z, which is bounded by $(k+1)!$. Since S can be chosen in at most $|V(H)|^{k+1}$ different ways, we have

$$|\Gamma_{\mathrm{f}}(H, z)| \leq 2^{\binom{k+1}{2}} \cdot (k+1)! \cdot |V(H)|^{k+1},$$

from which we get (iii) by noting that k is a constant. Finally, (iv) holds along the lines in the proof of (iii) above by noting that all cut vertices of H can be found in time $O\left(|V(H)| + |E(H)|\right)$ and it can be decided whether an injective function $\psi : S \to X_z$ is an induced subgraph isomorphism from $H[S]$ to $G[X_z]$ in constant time, as $|S|, |X_z| \leq k+1$. ■

3.4.3. Deciding Induced Subgraph Isomorphism

We summarize how to utilize feasible characteristics efficiently for deciding induced subgraph isomorphism. Let H be a strong candidate pattern generated by Algorithm 5 and G be a transaction graph, both of treewidth at most k. Furthermore, let *NTD*(G) be a nice tree-decomposition of G and r the root of *NTD*(G). By Lemma 3.1 and Theorem 3.6, $H \preccurlyeq_i G$ if and only if there is a feasible r-characteristic $(S, \mathcal{R}, K, \psi) \in \Gamma_{\text{f,ch}}(H, r)$ with $K = H$. The algorithm deciding $H \preccurlyeq_i G$ assumes that all nodes z in *NTD*(G) is associated with a set containing all elements of $\Gamma_{\text{f,ch}}(P, z)$, for all frequent patterns $P \in \mathcal{F}^{(H)} \setminus \{H\}$. It visits the nodes of *NTD*(G) in postorder traversal and calculates first $\Gamma_{\text{f}}(P, z)$ for all nodes z visited; this can be done in time polynomial in the size of H by (iv) of Theorem 3.6. It then computes $\Gamma_{\text{f,ch}}(P, z)$ by testing for all $\xi = (S, \mathcal{R}, K, \psi) \in \Gamma_{\text{f}}(P, z)$ whether ξ is a characteristic. Depending on the type of z, this test can be performed by checking the condition given in the corresponding case below:

> LEAF: By the case LEAF of Lemma 3.2, ξ is a characteristic iff $\mathcal{R} = \emptyset$.
>
> SEPARATOR: Let z' be the child of z in *NTD*(G) and let $\mathfrak{S}(\xi)$ be the set of all iso-quadruples $\xi' \in \Gamma(H, z')$ that satisfy conditions (S.a)–(S.c) of Lemma 3.2. Using similar arguments as in the proof of Lemma 16 in [HR10], one can show that (i) ξ is a characteristic if and only if $\Gamma_{\text{f,ch}}(H, z') \cap \mathfrak{S}(\xi) \neq \emptyset$ and (ii) $\mathfrak{S}(\xi) \subseteq \Gamma_{\text{f}}(H, z')$ and thus, it can be computed in time polynomial in the size of H.
>
> JOIN: Let z_1 and z_2 be the two children of z in *NTD*(G). To give the condition for this case, we need a definition. Let $\xi_i = (S_i, \mathcal{R}_i, K_i, \psi_i) \in \Gamma(P_i, z_i)$ for some $P_i \in \mathcal{F}^{(H)} \setminus \{H\}$ $(i = 1, 2)$. We assume without loss of generality that K, K_1, and K_2 are pairwise vertex disjoint. The *join* of ξ_1 and ξ_2 with respect to ξ, denoted by $\oplus_\xi(\xi_1, \xi_2)$, is an iso-quadruple $(S', \mathcal{R}_1 \cup \mathcal{R}_2, K', \psi')$ relative to z obtained from $(S_1 \cup S_2, \mathcal{R}_1 \cup \mathcal{R}_2, K_1 \cup K_2, \psi_1 \cup \psi_2)$ by (i) replacing u_1 and u_2 in $S_1 \cup S_2$, $K_1 \cup K_2$, and $\psi_1 \cup \psi_2$ with a new vertex u for all vertex pairs $u_1 \in S_1$ and $u_2 \in S_2$ with $\psi_1(u_1) = \psi_2(u_2)$ and by (ii) connecting in K' all original vertices $u, v \in S'$ with $u \in S_1$ and $v \in S_2$ by an edge labeled by ℓ if the vertices $u', v' \in S$ with $\psi(u') = \psi_1(u)$ and $\psi(v') = \psi_2(v)$ are connected in K by an edge labeled with ℓ. One can check that this definition is in fact an adaptation of conditions (J.a)–(J.c) of Lemma 3.2. In a way similar to the proof of Lemma 17 in [HR10], one can show that (i) ξ is a characteristic if and only if there are $\xi_i = (S_i, \mathcal{R}_i, K_i, \psi_i) \in \bigcup_{P \in \mathcal{F}^{(H)}} \Gamma_{\text{f,ch}}(P, z_i)$ for $i = 1, 2$ with $\xi \equiv \oplus_\xi(\xi_1, \xi_2)$ and that (ii) $\oplus_\xi(\xi_1, \xi_2)$ can be computed in time polynomial in the size of ξ, ξ_1, and ξ_2 for any ξ_1, ξ_2, implying that it can be decided in time polynomial in the size of $\mathcal{F}^{(H)}$, i.e., in *incremental polynomial time*, whether ξ is a characteristic.

Combining the arguments above with Lemma 3.1, we get Theorem 3.7 below for condition (v) of Theorem 3.2. Together with Theorems 3.4 and 3.5, this completes the proof of our main result stated in Theorem 3.3.

Theorem 3.7 Let $\mathcal{G}$ be the class of graphs of bounded treewidth. For every $H, G \in \mathcal{G}$ such that H is a strong candidate pattern generated by Algorithm 5, it can be decided in time polynomial in the combined size of the input $\mathcal{D}$ and the set of frequent patterns listed before H whether $H \preccurlyeq_i G$.

3.5. Summary

By the main result, the FCISM problem can be solved in incremental polynomial time for graphs of bounded treewidth. Positive results based on the dynamic programming framework suggest the investigation of further, computationally hard graph pattern matching operators for graphs of bounded treewidth (e.g., (induced) homeomorphism or (induced) minor embedding). We suspect that the systematic study of these and other graph pattern matching operators will result in an efficient *parameterized* frequent pattern mining algorithm for graphs of bounded treewidth, with the graph pattern matching operator as the parameter. Designing such a *generic* pattern mining algorithm is a very challenging project because, as the results in[HR10] and in this paper show, different graph pattern matching operators may require entirely different pattern refinement operators and entirely different combinatorial characterizations of feasible iso-quadruples.

The results of this paper raise interesting open problems. For example, it is an open question whether the positive result formulated in Theorem 3.3 can further be strengthened. In particular, can the FCISM problem be solved with *polynomial delay* for graphs of bounded treewidth? By setting the frequency threshold θ to 1, our main result implies that one can efficiently generate all *distinct* connected induced subgraphs of a graph of bounded treewidth. Does this positive result hold for arbitrary graphs as well? Or does the negative result given in Theorem 3.1 apply even to the special case that the database contains a single (arbitrary) graph and the frequency threshold is set to 1?

CHAPTER 4

Pattern Structure Analysis for Episodes

Outcomes of pattern mining would be applicable to many problems. That is, a list / set $\mathcal{F}$ generated can be regarded as *characteristic sets* (i.e., features) extracted from a given transaction database $\mathcal{D}$. This chapter particularly discusses the question on how to utilize the result set $\mathcal{F}$ in Knowledge Discovery.

The setting in this chapter is extracting patterns in the KDD process from indirectly generated graphs via mining (Example 2 in Chapter 1). We consider applying episode mining to sequential databases is an important activity to find valuable insights on the databases in the form of directed acyclic graphs. We assume that the result $\mathcal{F}$ can be sorted, and that we have a linear list of frequent episodes in support descending order. In practice, however, many episodes often subsume others, which has detrimental effects particularly when complex structures are considered, although efficient mining algorithms exist. We then need to consider how to utilize the list / set $\mathcal{F}$.

We here propose Pattern Structure Analysis (PSA) to build a compact view of enumerated episodes to support exploratory analysis by users from $\mathcal{F}$. PSA provides a way of obtaining clusters of episodes with their futures and of giving relations among the clusters. Both of them are represented by lattices compactly and then lattices can be applicable for utilizing enumerated episodes. In the design of PSA we adopt *star graphs* as a basic component of features, which are easy to interpret. By adopting the wildcard of symbols, which is easily introduced, PSA can merge similar episodes aggressively in clusters, which could be useful to get more insights from the mere linear list of all episodes. We show experimental results obtained using PSA and discuss an application of the obtained lattice structure for ranking clusters to show that our method effectively obtains informative ones from episodes enumerated.

keywords: Episode Pattern Mining, Generalized Pattern, Formal Concept Analysis, Pattern Structure, Ranking Concept, Exploratory Data Analysis

This chapter is based on the publications below:

- Otaki, K., Yamamoto, A.: Pattern Structures for Understanding Episodes, In Proc. of CLA 2014, CEUR Workshop Proceedings Vol.1252 (http://ceur-ws.org/Vol-1252/), pp.47-58, 2014.
- Otaki, K., Yamamoto, A.: Pattern Structure Analysis for Episode Mining, IEICE Transactions on Information and Systems (submitted, under the revision).

Note: This chapter is not open to the public because contents in this chapter is under the revision and the reviewing process as a journal paper. The main contents here are summarized as follows.

- Section 4.1 explains the introduction of problems in this chapter.
- Section 4.2 gives preliminaries to develop pattern structures for episodes based on Formal Concept Analysis.
- Section 4.3 defines pattern structures for epsidoes to make clusters of them.
- Section 4.4 provides experimental results using pattern structures for episodes.
- Section 4.5 summarizes related work.
- Section 4.6 concludes this chapter.

CHAPTER 5

MDL Principle for Pattern Set Mining on Lattices

Unsupervised pattern set mining is a problem focusing on post-processing of pattern mining, where a task is to select a subset of the whole output $\mathcal{F}$ of a pattern mining algorithm without any labels. From algorithmic viewpoints, it is related to problem of putting suitable and tractable constraints on the set $\mathcal{F}$ to make us possible to find a valuable subset. The closedness, one of the most fundamental constraints to filter $\mathcal{F}$ to get the set of closed patterns $\mathcal{CF} = \{X \in \mathcal{F} \mid X \text{ is closed}\} \subseteq \mathcal{F}$, is important conception in both FCA and pattern mining.

Although various efficient mining algorithms have been proposed (recall Chapter 2), finding characteristic and easy to interpret subsets of $\mathcal{F}$ is still challenging in the unsupervised setting. Out of various problems and methods studied in the field, this chapter focuses on methods using a *Information Theory-based criterion*, that is, the *Minimum Description Length* principle. In the field, MDL-based methods have been recognized as an effective and general approach in unsupervised pattern set mining. However, it requires us to design the MDL evaluation *from scratch* according to target patterns. We therefore investigate a new framework applicable to various patterns using lattices, which are beneficial to the MDL evaluation.

A key idea is revising an existing model for itemsets by using *edit operations* defined on lattices among concepts on them, which enables us to consider additional information such as background knowledge and helps us to design the MDL evaluation in general settings when lattices are given. We experiment our method to see that our proposal helps us to obtain informative results from the whole sets, and confirm that our model is applicable to various patterns.

keywords: Minimum Description Length Principle, Unsupervised Pattern Set Mining, Edit Operation, Lattice Structure

This chapter is based on the publication below:

• Otaki, K., Yamamoto, A.: Edit Operations on Lattices for MDL-based Pattern Summarization, In Proc. of ICFCA 2015 Supplementary, CEUR Workshop Proceedings Vol.1434, (http://ceur-ws.org/Vol-1434/, as FCA&A Workshop), pp.17-32, 2015.

Note: This chapter is not open to the public because contents in this chapter is before the submission to an international journal. The main contents here are summarized as follows.

- Section 5.1 explains the introduction of problems in this chapter.
- Section 5.2 gives preliminaries focusing on the problem of pattern set mining and the MDL principle in the pattern mining literature.
- Section 5.3 defines our new formalization based on the Two-part MDL principle and lattice structures of patterns to solve unsupervised pattern set mining.
- Section 5.4 provides examples of generalizations of coering based on lattice structures.
- Section 5.5 evaluates proposals via computational experiments using benchmark datasets.
- Section 5.6 concludes this chapter.

CHAPTER 6

Periodical Skeletonization for Periodic Pattern Mining

Finding periodic regularity from sequential databases is an important topic in Knowledge Discovery. In pattern mining, such regularity can be formulated as *periodic patterns*, where constraints using the *periodicity* of occurrences of symbols can be introduced based on the ordinal sequential pattern mining problem.

Although efficient enumeration algorithms have been studied in the literature, applying them to real databases is still challenging because they are noisy and sparse, and consequently most patterns are not extremely frequent. These phenomena cause a pattern (combinatorial) explosion and the difficulty of tuning a threshold parameter θ in that the number of frequent patterns is sensitive with θ. To overcome these issues we provide a novel pre-processing method called skeletonization by considering co-occurrences of symbols. The skeletonization is recently develop pre-processing method, for sequential patterns, aiming at finding clusters of symbols, aiming at shrinking the space of all possible patterns in order to avoid the combinatorial explosion. The key idea is to compute similarity within symbols in patterns from a given database based on the definition of patterns we would like to mine, and to use *graph clustering* methods based on the similarity computed. Although the original method cannot allow for periods, we generalize it by using the periodicity.

Experimental results using both synthetic and real datasets show the effectiveness of our approach, and compare results of periodic pattern mining with and without the skeletonization to see that our method is helpful for mining comprehensive (i.e., readable) patterns. Our main interests in the chapter are how effective pre-processing of pattern mining is, and how we can confirm the effects of pre-processing on pattern mining.

keywords: Periodic Pattern Mining, Pre-processing, Graph-based Clustering, Pattern Spectrum

This chapter is based on the publications below:

- Otaki, K., Yamamoto, A.: Periodical Skeletonization for Partially Periodic Pattern Mining, In Proc. of DS 2015, pp.186-200, 2015, DOI:10.1007/978-3-319-24282-8_16.
- Otaki, K., Yamamoto, A.: Periodic Pattern Mining with Periodical Co-occurrences of Symbols, IPSJ Transactions on Mathematical Modeling and Its Applications (TOM), vol.9(1), pp.32-42, 2016.

6.1. Introduction

Recall our target described in Example 2 in Chapter 1, where we assume that transactions in databases have timestamps as their auxiliary attribute and transactions are sorted in the chronological order. If such an order is important to a sequential database, typical periods related to clocks or calendars (e.g., hour, day, etc.) may contribute to hidden regularity in sequences as the order is related to time directly. Therefore assuming that such periodic behaviors may appear in various sequential databases (e.g., trajectory, life-log) is natural in data mining. To obtain valuable but hidden insights from databases by capturing periodic regularity, *periodic pattern mining* have been studied[HDY99,HGY98,YHCL13] and applied to various problems[HLXC12,LG13].

Periodic Pattern Mining

We have several variations on the definition of periodic patterns. The fundamental ones are *full periodic patterns* and *partial periodic patterns*[HDY99]. Note that we now assume that patterns are sequences of symbols drawn from a given base-set $\mathcal{B}$ for the sake of simplicity.

Full Periodic Pattern

Partial Periodic Pattern

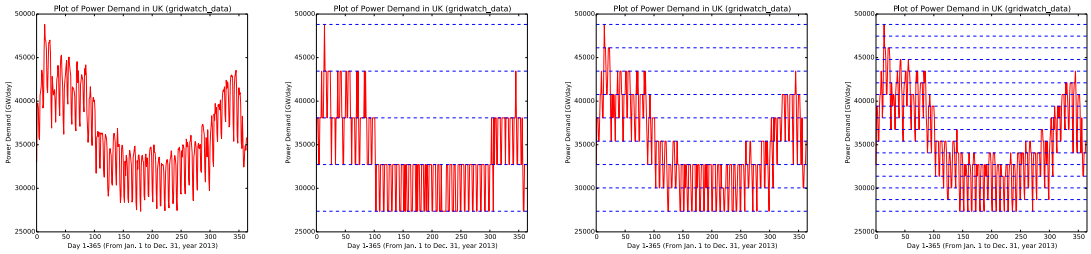
For example, let $\mathcal{B} = \{\text{sns}, \text{news}, \text{blog}, \text{shops}\}$. We consider a sequence s_{user}:

$$s_{\text{user}} = (\text{sns}, \text{news}, \text{blog}, \text{sns}, \text{news}, \text{blog}, \text{sns}, \text{shop}, \text{blog})$$

representing a log of categories of Web sites visited by a user. A pattern in s_{user} $(\text{sns}, \text{news}, \text{blog})$ appears twice, and this is called *full periodic pattern* of period length 3. Full periodic patterns require that all symbols should be fully specified. In some cases, such requirement is so strong and it is difficult to handle various periodic behaviors. As more flexible patterns, *partial periodic patterns* have been studied[HGY98]. For example, a partial periodic pattern $(\text{sns}, \perp_{\mathcal{B}}, \text{blog})$ appears 3 times, where $\perp_{\mathcal{B}}$ is the wildcard symbol of length 1 representing any symbol in $\mathcal{B}$. As partial periodic patterns can contain the symbol $\perp_{\mathcal{B}}$, they are more flexible than full periodic patterns to capture periodic behaviors in databases. In mining these periodic patterns, we assume that a given sequence s is divided into $\lceil \frac{|s|}{\text{P}} \rceil$ fragments, where P is a period of users' interest, and the fragments are used to evaluate patterns: In the example above, the pattern $(\text{sns}, \perp_{\mathcal{B}}, \text{blog})$ appears 3 times in fragments $(\text{sns}, \text{news}, \text{blog})$, $(\text{sns}, \text{news}, \text{blog})$, and $(\text{sns}, \text{shop}, \text{blog})$ of s_{user}.

Although many efficient algorithms have been developed[HDY99,YHCL13] for the enumeration, it is still challenging to use them in practice because the number of enumerated patterns highly depends on the number $|\mathcal{B}|$ of symbols we use. When databases get large, $|\mathcal{B}|$ increases as well and databases should contain more noise. This fact consequently makes evaluating patterns by their support (i.e., the number of occurrences) difficult because the supports of most patterns appear is similar and relatively small. That is, the space of (frequent) patterns on $\mathcal{B}$ get *sparse* with respect to the space of all possible patterns[*].

[*] Consider to find all partially periodic patterns up to the length k on $\mathcal{B}$. Let $\mathcal{B}_{\perp_{\mathcal{B}}} = \mathcal{B} \cup \{\perp_{\mathcal{B}}\}$. All possible combinations are in $\mathcal{B}_{\perp_{\mathcal{B}}} \cup \mathcal{B}^2_{\perp_{\mathcal{B}}} \cup \cdots \cup \mathcal{B}^k_{\perp_{\mathcal{B}}}$, which can become much larger than that of all patterns appearing in databases in practice.



(a) Original (b) $|\mathcal{B}| = 4$ (c) $|\mathcal{B}| = 8$ (d) $|\mathcal{B}| = 16$

Figure 6.1.: A numeric sequence of electric power demand in UK, 2013 in Figure 6.1a, and its discretization with $4, 8, 16$ symbols (Figure 6.1b, 6.1c, and 6.1d).

Motivating Examples For both numerical (e.g., price, temperature) and symbolic (e.g., item, product) sequences, preparing a large set $\mathcal{B}$ of symbols is essential to achieve the high resolution of describing phenomena. For example, Figure 6.1a shows a sequence of electric power demand per day in UK, 2013. To deal with the sequence in pattern mining, we discretize the sequence with dividing values into $|\mathcal{B}|$ bins uniformly[†] as seen in Figure 6.1d with $|\mathcal{B}| = 16$. Clearly, we can represent a sequence as a symbolic sequence with a smaller loss with a larger set $\mathcal{B}$ by comparing figures (in Figures 6.1b, 6.1c, and 6.1d). In Figure 6.1d, however, only a few combinations of $\mathcal{B}$ appear consecutively. It is difficult to tune the set $\mathcal{B}$ while taking a balance among the expressiveness and the sparseness. Now a typical periodic behavior is that *the demand gets higher every weekend*, which could be obtained by frequent patterns, where symbols corresponding to low values are followed by those doing to high values. We believe that such high-level patterns are more informative and useful to analyzing databases in Knowledge Discovery.

We also consider those stored from LAST.FM[‡], as another example, which are sequences of songs listened by users. We take some logs of users from an open dataset (See Section 3 of[Cel10]), where a sequence $s = \langle S_1, S_2, \dots \rangle$ is a log of a user and each S_i is the set of songs heard in the index i in which the index i corresponds to a 1 hour interval of the log (e.g., the set S_4 shows the listened songs during 0 a.m. to 1 a.m.). For example, the sequence for user ID 808 is length $16,913$ log, where the user listened to $24,310$ songs and $1,340$ out of $16,913$ intervals are not empty (i.e., in other intervals the user did not listen to any songs). Then if we would like to analyze some daily behaviors (i.e., P $= 24$) including the empty situation, $24,310^{24}$ is the upper bound of the size of all possible combinations. This is intractable and data we have are obviously sparse.

Approaches In pattern mining, therefore, Liu et al.[LZX+14] and others insisted that users need to tune the set $\mathcal{B}$ carefully, and proposed the temporal skeletonization for symbolic sequential patterns. Their idea is to construct clusters of symbols and

[†]If the range of values $[0, 10)$ and $|\mathcal{B}| = 4$, values in $[0, 10]$ would be categorized into either $[0, 2.5), [2.5, 5.0), [5.0, 7.5)$, or $[7.5, 10)$, and symbolic alphabets are assigned to encode the sequence into a symbolic sequence.

[‡]http://last.fm/

assign each cluster a label. Then a sequence can be translated into a high-level and potentially comprehensive sequences of the labels. By grouping symbols into clusters, we can reduce the size $|\mathcal{B}|$. We develop such method for periodic analyses by generalizing the idea[LZX+14], and discuss frequently occurring high-level patterns with the periodicity. Note that such idea to reduce the size $|\mathcal{B}|$ is sometimes studied as *alphabet indexing* in machine learning (e.g.,[SSS+93]). In the existing method[LZX+14], to represent comprehensive sequential patterns, *left-to-right* HMMs are *implicitly* assumed as their models. Based on the study we take into account the periodicity of the given period length P by considering a bit different models of sequences, i.e., *cyclic HMMs*, which can be applicable to describe periodical behaviors of sequences comprehensively.

Left-to-right HMM

Cyclic HMM

In addition to that, we would like to emphasis on the fact that many existing studies dealt with DNA sequences, which requires only 4 symbols (i.e., $\mathcal{B} = \{\mathrm{T}, \mathrm{C}, \mathrm{A}, \mathrm{G}\}$). In such a situation, the combinatorial explosion is only depending on the length l of patterns we try to mine. However, this situation is a bit restricted as many sequences require more symbols in general. Therefore developing a new approach with the periodicity is an important remained problem. We thus focus on the point by following the existing method[LZX+14]. Since the temporal skeletonization cannot be applied to periodic settings, we generalize it by using the idea of periodic extensions of functions.

6.2. Periodic Patterns and Skeletonization

Let $\mathcal{B}$ be the alphabet. The set $\mathcal{B}^*$ denotes the Kleene closure of $\mathcal{B}$. We use $\mathcal{B}^+ \equiv \mathcal{B}^\star \setminus \{\epsilon\}$, where ϵ is the empty string. For a sequence $s \in \mathcal{B}^+$, $|s|$ denotes the length of s. We let $|\epsilon| = 0$. In addition, s_i and $s_{i,j}$ represent i-th element and the continuous subsequence from i to j of s ($i < j$), respectively. Let P be a fixed integer representing the period of users' interest. Without a loss of generality, we generalize the notations above for the case of sequences $s \in (2^{\mathcal{B}})^+$ of sets of symbols.

6.2.1. Frequent Partially Periodic Pattern Mining

Periodic behaviors of databases can be modeled as *partially periodic patterns*. In the literature, transaction databases can be defined as (singleton) sets of sequences on $\mathcal{B}$ or those of subsets of $\mathcal{B}$. Then patterns are intuitively sequences of (sets of) symbols. An important concept used is *periodic segments* of sequences.

Periodic Segment

Definition 6.1 (Periodic Segment) For an *event sequence* $s \in \mathcal{B}^+$ and a period P, s can be divided into m $(= \lceil \frac{|s|}{\mathrm{P}} \rceil)$ *mutually disjoint segments*, denoted by $s = \langle ps_1, ps_2, \ldots, ps_m \rangle$, where for $1 \leq i \leq m$, $ps_i = s_{(i-1)m, im-1}$.

A periodic pattern can be a mere sequential pattern in the definition of the pattern language $\mathcal{L}$. The difference is only in the existence of the wildcard $\perp_{\mathcal{B}}$ to allow patterns partially fixed with characters by $\mathcal{B}$.

Definition 6.2 (Partial Pattern) A sequence from $\mathcal{B} \cup \{\perp_{\mathcal{B}}\}$ is a (*partial*) *pattern*, where the special character $\perp_{\mathcal{B}} \notin \mathcal{B}$ represents any event of length 1. Partial Pattern

We are now interested in partial patterns appearing in periodic segments frequently on a given sequence s. Then, for a sequence s and a pattern p, it is necessary to evaluate whether or not p has an interesting occurrence in s. The traditional measure for the purpose is to adopt the *support* of partial patterns.

Definition 6.3 (Support) The *support* of a pattern p, denoted by $\mathrm{Sup}_{\mathrm{P}}(p)$, is defined as $\mathrm{Sup}_{\mathrm{P}}(p, s) = |\{ps_i \mid s = \langle ps_1, \ldots, ps_m\rangle, ps_i \preceq p\}|$, where $ps_i \preceq p$ means that for all $1 \leq i \leq |p|$ it satisfies either $p_i = \perp_{\mathcal{B}}$ or $p_i = s_i$. We say that a pattern p is *frequent* if $\mathrm{Sup}_{\mathrm{P}}(p) \geq \theta$ for a user-specific threshold θ.

Such a pattern p is called a partially periodic pattern (PPP) when the period P is given and important. Partially Periodic Pattern

Example 6.1 (Support) For $s = abcabdabb = \langle abc, abd, abb\rangle$ and the a constant (period) $\mathrm{P} = 3$, $\mathrm{Sup}_{\mathrm{P}}(ab\perp_{\mathcal{B}}, s) = \frac{3}{3} = 1.0$.

Now the mining problem is formally described below.

Problem 6.1 (The FPPPM problem) Let θ be a user-specific threshold and P a given period length. For a sequence s, the FPPPM problem is defined as to list all partially periodic patterns p from s satisfying $\mathrm{Sup}_{\mathrm{P}}(p) \geq \theta$. FPPPM Problem

Several efficient algorithms have been developed: Han *et al.* showed a fundamental algorithm using max sub-pattern trees[HGY98]. Algorithms using max sub-pattern trees are used frequently in several application problems using periodic patterns[HLXC12,LG13]. Yang *et al.* proposed to use tuple representations for periodic patterns and a depth-first search algorithm based on the PrefixSpan[PHMA+04] used in sequential pattern mining[YHCL13].

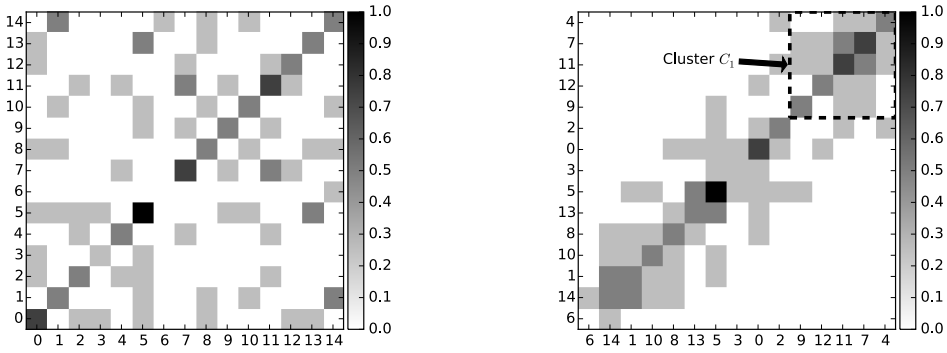
6.2.2. Temporal Skeletonization

We review the original definition of *temporal graphs* to explain the idea of the temporal skeletonization in[LZX+14], which tries to build a *similarity graph*[§] from a given database $\mathcal{D}$ to capture the similarity within symbols in $\mathcal{B}$. Note that in this literature, a transaction in $\mathcal{D}$ is defined as a sequence on $\mathcal{B}$. Temporal Graph Skeletonization

Definition 6.4 (Temporal Graph[LZX+14]) Let $G = (V, E)$ be a weighted undirected graph and $\mathcal{D} = \{s^{(1)}, \ldots, s^{(N)}\}$ be the set of sequences of symbols from $\mathcal{B}$. For two symbols $x, y \in \mathcal{B}$, the weight $W_{x,y}$ of the edge corresponding to $\{x, y\}$ is defined as Temporal Graph

$$W_{x,y} = \frac{1}{N} \sum_{n=1}^{N} \sum_{\substack{1 \leq i \leq j \leq |s^{(n)}|, \\ |i-j| \leq r}} \mathbb{1}[s_i^{(n)} = x \wedge s_j^{(n)} = y] \tag{6.1}$$

[§]A *similarity graph* is a weighted graph in which vertices represent data points and edges represent the similarity between two points with their weights.



(a) Computed $W_{s,t}$.

(b) Re-ordered $W'_{s,t}$.

Figure 6.2.: Compute two heatmaps from a toy example in[LZX+14]: Figure 6.2a shows the original W computed from $\mathcal{D}$, and Figure 6.2b is the re-ordered one, in which a cluster C_1 is drawn.

where N is the number of sequences, r be the *window width*, $\mathbb{1}[p]$ is the indicator function that returns 1 if and only if the predicate p is true; 0 otherwise.

Intuitively we count the number of *co-occurrences* of symbols x and y in a window. The indicator function $\mathbb{1}[\cdot]$ can be replaced with other similarity measures; The authors in[LZX+14] used the exponential function $\exp(-k|i-j|)$ with a parameter k. In constructing a temporal graph G, all indices of sequences in $\mathcal{D}$ are taken into account several times. After constructing G, users try to find clusters of symbols by applying clustering methods to G (e.g., *spectral clustering*[NJW01,SM97]). The problem of finding clusters can be formulated as a standard graph-based optimization problem with some constraints as explained in[LZX+14], where an important step is to compute eigenvalues and eigenvectors from G. Now a computed matrix W of weights can be represented as a heatmap illustrated in Figure 6.2.

Spectral Clustering

Example 6.2 (Temporal Skeletonization) Let $\mathcal{D} = \{s^{(1)}, s^{(2)}, s^{(3)}, s^{(4)}\}$, where $s^{(1)} = \langle 12, 7, 9, 5, 3, 0, 8, 10, 1\rangle$, $s^{(2)} = \langle 9, 11, 12, 0, 13, 5, 1, 14, 6\rangle$, $s^{(3)} = \langle 4, 7, 11, 2, 5, 7, 9, 1, 14\rangle$, and $s^{(4)} = \langle 7, 11, 4, 2, 0, 7, 10, 14, 8\rangle$ on $\mathcal{B} = \{0, 1, 2, \ldots, 15\}$. We compute the weights and represent them by a heatmap, where both the x-axis and y-axis correspond to the order of the alphabet $\mathcal{B}$. That is, on some (i, j), the thickness in the heatmap corresponds the value $W_{\mathcal{B}_i,\mathcal{B}_j}$. After applying the spectral clustering, we can re-order indices of W as shown in Figure 6.2b. For example, we can find a cluster of symbols such as $C_1 = \{4, 7, 9, 11, 12\}$, which is the upper right area in Figure 6.2b. Note that C_1 appears in prefixes of sequences in $\mathcal{D}$. Then we can now conjecture that all sequences in $\mathcal{D}$ are in the form $(C_1, C_1, C_1, \ldots)$. This prefix consisting of cluster labels can be regarded as a *high-level* pattern of the sequences.

Models and Assumptions This method is based on an implicit assumption; *symbols x, y appearing closely and temporally may belong to some meaningful cluster*. In

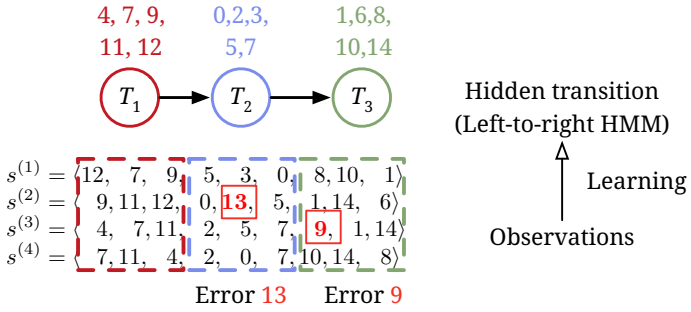


Figure 6.3.: Temporal clusters and their transition; assuming that a left-to-right HMM consisting of three states T_1, T_2, T_3, which represent hidden meaningful clusters behind sequences $s^{(i)}$, and learning them from the observations.

this scenario, a typical (left-to-right) HMM is adopted, illustrated in Figure 6.3, and based on the implicit assumption above, the method tries to find such clusters by using a moving window of width r and graph clustering methods. Left-to-right HMM

6.3. Periodical Skeletonization

The key idea for taking into account periodic information is simple: Extending functions representing areas that we check in computing weights to some periodic functions of the periodicity P of our interest. In order to analyze some monthly behaviors, for example, we set $\mathrm{P} = 31$.

The (sliding) *window* of width r used in the temporal skeletonization can be modeled by a *rectangular function* with width r and the origin i[¶]. By modifying this function in a periodic manner, we can deal with the periodicity of occurrences of symbols. We can easily imagine such techniques on the analogy of Fourier series and Fourier transforms. Recall the toy examples in Section 6.2.1. For an input sequence $s = abcabdabb$ and $\mathrm{P} = 3$, a frequent partially periodic pattern $ab\perp_{\mathcal{B}}$ appears 3 times in every segment abc, abd, and abb. This means that not only neighbors according to the sliding window $\mathrm{Rect}_{i,r}(\cdot)$, but also periodic information from i, that is, $i + \mathrm{P}, i + 2\mathrm{P}, i + 3\mathrm{P}, \ldots$ could be used to search for similar intervals. Window

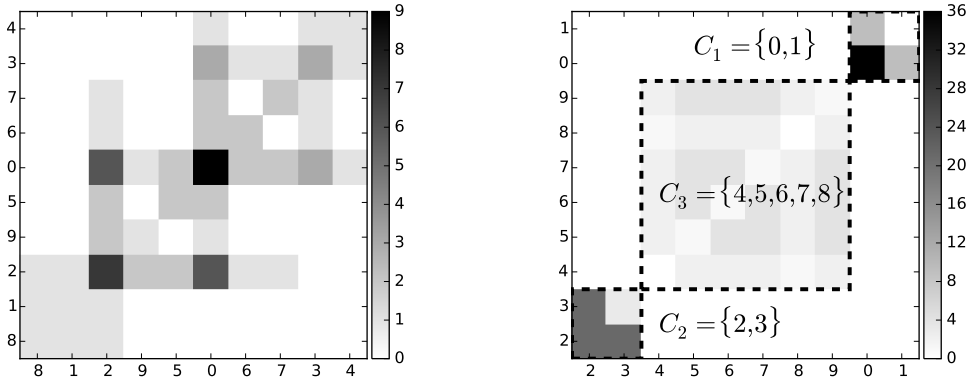
The above observation inspired our method, *periodical skeletonization*, where a similarity graph named a *periodic graph* is computed by observing the *periodic co-occurrences* of symbols from an input sequence s. Periodical Skeletonization

Definition 6.5 (Periodic Graph) Let $G = (V, E)$ be a similarity graph. The weights of edges in G from an input sequence s and a period P for two symbols x, y are computed as: Periodic Graph

$$W_{x,y} = \sum_{\substack{1 \le i,j \le |s| \\ \textbf{if } s_i = x \wedge s_j = y}} \mathbb{1}[|i - j| \le r] + \mathbb{1}[i \equiv j \pmod{\mathrm{P}}] \tag{6.2}$$

The second term is newly introduced based on the periodic information.

[¶]It is defined as $\mathrm{Rect}_{i,r}(t) = 0$ if $|t - i| > r$, 1 otherwise.



(a) By the temporal method

(b) By the periodical method

Figure 6.4.: An example of *periodic co-occurrences* of a sequence $s = \langle 0, 2, 6, 0, 2, 4, 0, 3, 7, \ldots, \rangle$. Figure 6.4a is a result only using the temporal information and Figure 6.4b is that adopting the periodic information only, where rectangles are the discovered clusters.

We can also replace the right-hand side of Equation 6.2 with similarity functions by adapting them in a similar fashion based on the temporal skeletonization.

Example 6.3 (Periodical Skeletonization)] Figure 6.4 illustrates examples of computing Equation 6.2 from $s = \langle 0, 2, 6, 0, 2, 4, \ldots \rangle$ with $\mathcal{B} = \mathbb{N}$. Figure 6.4a is computed by the temporal skeletonization, while Figure 6.4b adopts the periodic term only in Equation 6.2. We can see 3 clusters as rectangles: $C_1 = \{0, 1\}, C_2 = \{2, 3\}$ and $C_3 = \{4, 5, 6, 7, 8\}$ in Figure 6.4b, and they are clear than those in Figure 6.4a (it is hard to find clusters from it).

Models and Assumptions Our basic observation is that we can represent periodic sequences by *cyclic HMMs*. From the definition of patterns, we assume that symbols appearing frequently *and periodically* in s should belong a meaningful cluster in periodic data analyses with period P. Our computation is then a bit generalized from the existing study, where the *periodical co-occurrences of symbols* in s are newly taken into account.

An example is given in Figure 6.5. For example, to simulate a partially periodic pattern $02\perp_{\mathcal{B}}$, in T_1 and T_2, $\mathcal{H}$ outputs 0 and 2, respectively with high probability $100\times(1-u)$% and outputs 1 and 3 with low probability $100\times u$%. On the other hand in T_3, $\mathcal{H}$ generates $\{4, 5, 6, 7, 8, 9\}$ uniformly. By generating sequences of length N from $\mathcal{H}$, we can obtain a sequence s containing the partially periodic pattern $02\perp_{\mathcal{B}}$ very frequently. Compared with the result in Figure 6.4a, we can confirm that C_1, C_2, C_3 correspond to T_1, T_2, T_3 are found by the skeletonization more clearly as blocks in the matrix in Figure 6.4b.

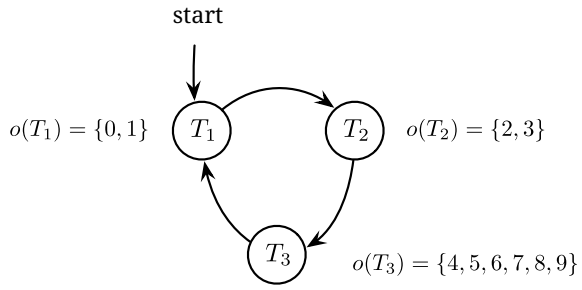


Figure 6.5.: Settings of HMMs used for Example 6.3. The output symbols from states T_1, T_2, and T_3 are set to be $o(T_1) = \{0, 1\}$, $o(T_2) = \{2, 3\}$, and $o(T_3) = \{4, 5, 6, 7, 8, 9\}$. The two symbols 0 and 2 are generative with a high probability $1 - u$ and 1 and 3 are done with a low probability u $(0 < u \leq 0.25)$. Symbols from T_3 is generated uniformly.

6.3.1. Set Sequences

Let us reminder the computation of W_{s_i,s_j} when indices i and j are considered in $s \in \mathcal{D}$. The discussion above and the existing study[LZX+14] only consider the case when $s \in \mathcal{B}^+$. However, some applications of sequential and periodic pattern mining assume that s is a sequence of subsets of $\mathcal{B}$: $s \in (2^{\mathcal{B}})^+$. To deal with *set sequences* we consider two naïve methods below:

Set Sequence

SUM-UP for two sets S_i and S_j of indices i and j, compute straightforwardly weights and sum up them. That is, compute all $W_{S_{i,k},S_{j,l}}$ for $S_{i,k} \in S_i$ and $S_{j,l} \in S_l$, $S_{j,l} \in S_j$ in a pair-wise manner, and use them.

AVERAGE for two sets S_i and S_j, compute the weights in a pair-wise manner as well, and divide the weights $W_{S_{i,k},S_{j,l}}$ with $|S_i| + |S_j|$, to take an average contribution of each symbols in S_i and S_j.

By these small generalizations, we note that the skeletonization techniques can be generalized to set sequences. If not otherwise stated, we adopt the sum-up idea.

6.4. Experiments

We report experiments with synthetic and real datasets which should have simple periodic behaviors to observe the effect of our proposal. We also discuss the difference between two skeletonization methods.

6.4.1. Datasets

Now in this part, we use both synthetic and real datasets. The summary of these datasets is shown in Table 6.1. A synthetic dataset is generated by using the HMM shown in Figure 6.5. A real dataset, named PowerDemand, is a set of sequences of electric power demand in 2013, extracted from the GridWatch system[‖], which were previously used in Figure 6.1. The original sequence records power demand in UK

[‖]http://www.gridwatch.templar.co.uk/

Table 6.1.: Summary of datasets used in experiments.

Name	Length	$\lvert\mathcal{B}\rvert$	P	Note
HMM-600-u	600	10	3	with $u = 0.25$
PD-32	365	32	7	Discretized with level 32
PD-128F	100	128	7	Subset with level 128
Kyoto	43,833	359	365	The resolution 0.1 celsius

12 times per hour, that is, roughly 300 times per day. We take the simple average of them to construct a hourly sequence of power demand in 2013, named PD-32. Because an yearly record may contain many periodic behaviors (e.g., daily, weekly, monthly, etc.), we extract a small subset, named PD-128F, of PD-32 and make the resolution of $\mathcal{B}$ more clear by increasing the size $\mathcal{B}$ from 32 to 128 and taking a part roughly from summer to autumn. For PD-128F, we expect that the sequence have the period P $= 7$. As another real dataset, we use Kyoto, a sequence of the daily temperatures from 1880 to 2014 with P $= 365$ and $|\mathcal{B}| = 359$.

6.4.2. Preparations

We implemented the periodic skeletonization part in C++ [**], and apply the spectral clustering algorithm (and k-means algorithm in it) by using a built-in implementation by SCIKIT-LEARN[PVG+11] based on Python 2.7.8. All experiments are run on a machine of Mac OS X 10.10 with 2×2.26 GHz Quad-Core Intel Xeon processors and 64GB memory.

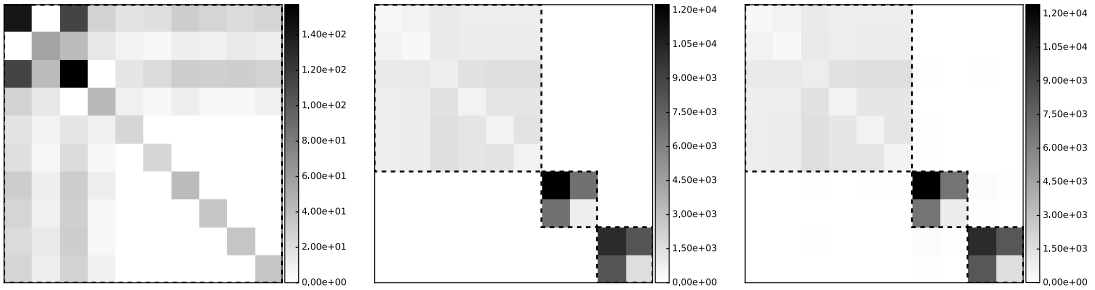
We would like to show computed graphs and the discovered clusters by the spectral clustering algorithm via temporal/periodic graphs. We set k by using the heuristic of the spectral clustering (Please see [VL07]), or by a small number (2 or 3, for example). In experiments we basically use only the original definition, i.e., we only use the delta function by the indicator function $\mathbb{1}[f]$. In the following, we prepare the following labels to represent methods: 1) DT means the temporal skeletonization, 2) DP users the periodic information only, and 3) DTP adopts the both of them.

6.4.3. Results

Out of several parameter settings we tried, we took a part of results to compare our periodic skeletonization with the temporal one. We showed results of synthetic data in Figure 6.6, and those of real datasets in Figures 6.7 and 6.8 with varying methods of computing weights, where the labels $\{\mathrm{DT}, \mathrm{DP}, \mathrm{DTP}\}$ represent the methods used.

Synthetic Datasets From results using synthetic data, we can conjecture that periodic information of temporal graphs are helpful to find clusters of symbols compared with Figure 6.6a and Figures 6.6b and 6.6c, where we would like to extract

[**] GCC 4.7 with -STD=C++11 without parallelization



(a) DT (b) DP (c) DTP

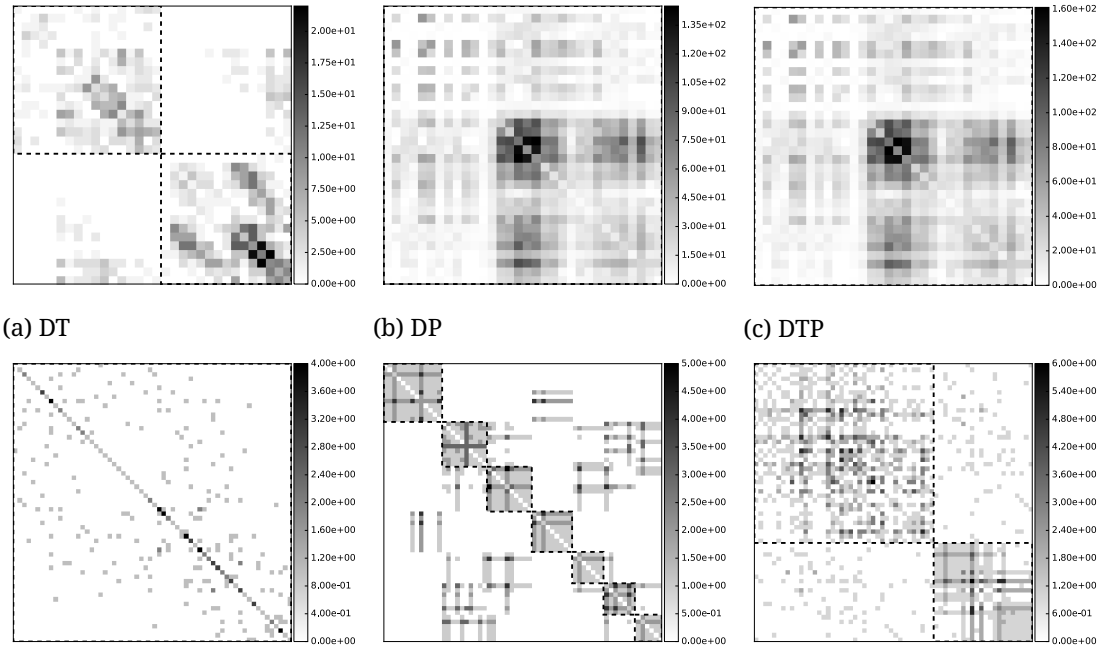
Figure 6.6.: Heatmaps representing similarity matrices of graphs from the synthetic sequence with $\mathrm{P} = 3$ and $r = 3$. Figures 6.6b and 6.6c successfully show clusters as *rectangles*.

periodic clusters, that is, clusters representing $\{0, 1\}$ and $\{2, 3\}$, which corresponds to T_1 and T_2 in the HMM in Figure 6.5, respectively. From the result using only temporal information in Figure 6.6a, however, we *cannot* find them. On the another hand, results using periodic information seen in Figure 6.6b and both of them in Figure 6.6c show two clusters $\{0, 1\}$ and $\{2, 3\}$ much clearly.

Real Datasets Results from real datasets should be affected by properties of sequences and the periodicity of them. In two cases with PD-32 and PD-128F, for example, results were symmetric with respect to methods: If we use the periodic information in Figures 6.7b and 6.7c, we cannot find any clusters but in Figures 6.7e and 6.7f, we can find a few clusters of symbols, which are similar to results of synthetic data. We guessed that the difference between PD-32 and PD-128F is whether or not there exists many periodic behaviors in sequences. Because we selected a subsequence from PD-32 as PD-128F to remove multiple periodic information, the periodic skeletonization with a fixed period parameter $\mathrm{P} = 7$ seemed to work well.

In results from Kyoto, we can see that there exist roughly 3 clusters in all outputs in Figure 6.8. If we adopt the periodic information, those clusters are also emphasized on visualized temporal graphs. For example, by comparing results in Figure 6.8a and Figure 6.8c, we confirm that two dense clusters (top left and bottom right) in Figure 6.8c are much more clear than in those Figure 6.8a. Note that these clusters are related to winter and summer, respectively. We conjecture that these visualized results are helpful to analyze given sequential databases and enumerated patterns, particularly when we need to run methods many times to tune parameters.

Conclusions We conclude experiments using sequences containing clear periodic behaviors. Originally, results of clustering symbols are sensitive to the definition of similarities. The previous study reported in [LZX+14] that results of the skeletonization seemed to be stable. As far as we investigated in experiments, with respect to the parameter r, which control a kind of smoothing of sequences, the results could be stable as well. We also see that our method could be helpful to *highlight periodic behaviors of sequences*. We guess that this result is also affected from the multiple



(a) DT (b) DP (c) DTP

(d) DT (e) DP (f) DTP

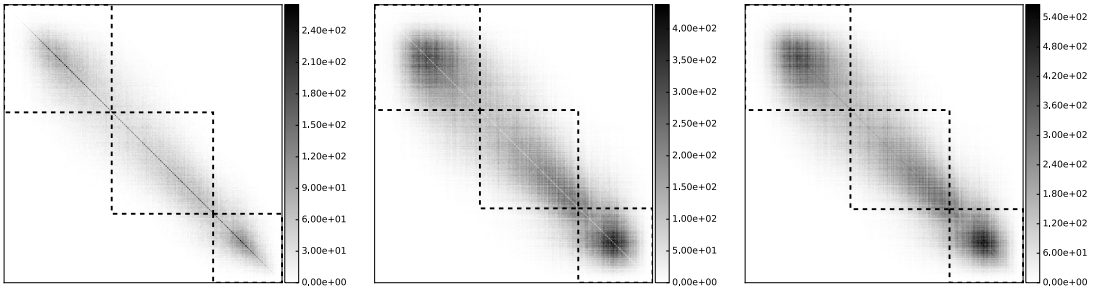
Figure 6.7.: Heatmaps from PD-32 (top row) and PD-128F (bottom row) with DT, DP, and DTP.

periodicity, and conclude that the periodic skeletonization help us to find underlying structures. Although the method sometimes (as seen in PD-32) disturbs results, it seems to work as we expected particularly when the periodicity is clear.

6.4.4. Unstable Case

As we show in Figure 6.5, the model behind our skeletonization procedure is a cyclic HMM. This assumption suits the FPPPM problem with (fixed) periodic segments. However, in general, the period of a sequence s is unstable because of noisy symbols or outliers. In the pattern mining part, mining algorithms allowing for a small gap (i.e., a kind of wildcard) between segments have been studied [ZKCY07]. Therefore we would like to discuss the same point on the skeletonization techniques. Because temporal clusters correspond to a kind of moving average on symbols, we conjecture that adopting both the temporal skeletonization and the periodic one should be useful to deal with such a situation.

To investigate this issue, we generated synthetic sequences from a cyclic HMM of P states, and see whether or not we can discover P clusters from the data. The results by DT (only temporal), DP (only periodic), and DTP (both) of the transition failure percentage $\eta = 20\%$ in Figure 6.9 were summarized in Table 6.2. We can confirm that, at least for synthetic data, periodic information used in the skeletonization is helpful to estimate hidden periodic clusters of sequences. From experiments, we guess that using both information could be helpful when real sequences are processed.



(a) DT (b) DP (c) DTP

Figure 6.8.: Heatmaps from Kyoto with DT, DP, and DTP.

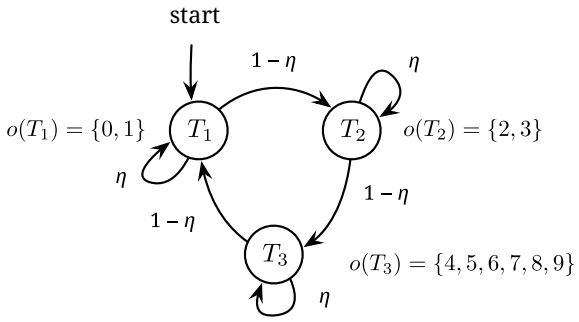


Figure 6.9.: A cyclic HMM including unstable periods with transition failures with probability η of the case $\mathrm{P} = 3$.

6.4.5. Case Studies

We provide results using (more) real datasets. One is from Last.fm data that we have used in Section 6.1. In addition to that, we here adopt 2-dimensional sequences (i.e., X and Y) representing trajectories of movements, and encode them as (1-dim) symbolic sequences for experiments.

Last.fm Datasets Because properties of data vary according to users in the service, we would like to investigate how results get for real datasets. Datasets are obtained from [Cel10] by gathering and ordering the logs of songs listened by users based on focusing the granularity "hour". One database corresponds to one sequence of sets of symbols (i.e., songs) by one user. For experiments, we take randomly users from the whole dataset, obtain sequences of sets of symbols, and use small parts of such sequences. We provide statistics of selected parts of sequences chosen by our method in Table 6.3. For a sequence $s = \langle S_1, S_2, \ldots, S_M \rangle$ of $S_i \subseteq \mathcal{B}$, L_{all} means $|s| = M$, L_{ne} shows $|\{S_i \mid S_i \neq \emptyset, S_i \text{ is in } s\}|$, $|\mathcal{B}|$ means the size of the set $|S_1 \cup \cdots \cup S_M|$, and $||\mathcal{S}|| = \sum_i |S_i|$, respectively. We set $\mathrm{P} = 24$ to analyze hourly behaviors.

We show results in Figures 6.10 and 6.11. Here we do not want to say which clustering results are good (or bad). From experiments by periodic information in the skeletonization we can confirm two kind of results: A type increases the number of clusters compared with the ordinal temporal skeletonization (e.g., from Figure 6.10a to Figure 6.10c). Another type, in contrast, decreases the number of

Table 6.2.: From P states cyclic HMM, for each state we generate 5 symbols uniformly. The transition failure η is 20%. Values show accuracy of recovering hidden clusters.

P	DT	DP	DTP
3	0.33	0.80	0.80
5	0.24	0.92	0.92

Table 6.3.: Statistics of user logs from the Last.fm dataset.

User ID	$\lvert L_{\text{all}}\rvert$	$\lvert L_{\text{ne}}\rvert$ (non-empty)	$\lvert\mathcal{B}\rvert$	$\lvert\lvert\mathcal{S}\rvert\rvert$
User 672	384	147	247	2,329
User 808	529	147	578	2,108

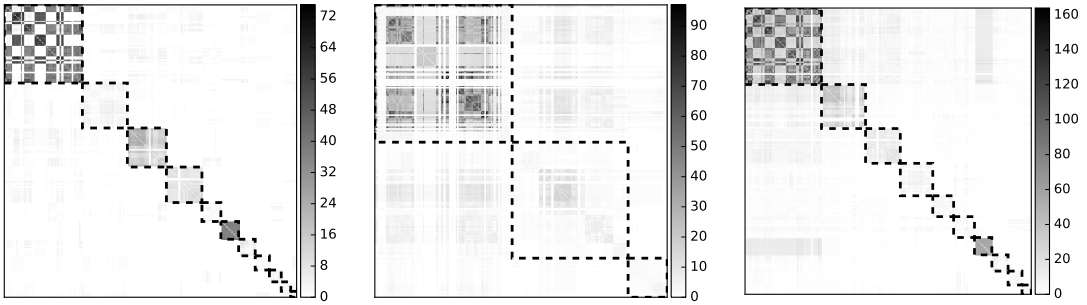
clusters (e.g., from Figure 6.10a to Figure 6.10b, Figure 6.11a to Figures 6.11b and 6.11c). As the periodic information help us to consider periodic co-occurrences of symbols, if there exist some periodic behaviors of sequences, then applying the periodicity skeletonization should be helpful. We can only confirm that in some cases the clustering work for our purpose. We conjecture that for some databases our method does not work as they contain no periodic regularity.

Trajectory Datasets As an another example, we adopted trajectory (timestamped) databases used in YZZ+10,YZXS11††. A trajectory here is an ordered sequence of pairs, i.e., (X, Y) corresponding to longitude and latitude of an entity. Its movements is regularly recorded and stored as a sequential database. An example of trajectory can be plotted in 2-dimensional space as shown in Figure 6.12. Note that a sequential database $\mathcal{D}$ is now encoded as a single sequence s by discretizing X and Y, and putting some integer $n \in \mathbb{N}$ on each grid. Table 6.4 shows some statistics of trajectories used. All trajectories are discretized with level $d = 256$, and many (X, Y) slots on the grid could be empty. Thus the table also shows the number of non-empty (X, Y) slots in discretized sequences. As they include some obvious outliers or noise, such data are cleaned by checking the mean and standard decreases of X and Y.

In contrast to the datasets used above, for trajectories, we have no idea on what are its period. In the following experiments, therefore, we provide a kind of exploratory data analyzing processes using skeletonization by observing the similarity graphs constructed. In addition, we only try the case $k = 2$, i.e., try to divide all symbols $\mathcal{B}$ into two clusters as the basis of recursively decomposition of $\mathcal{B}$.

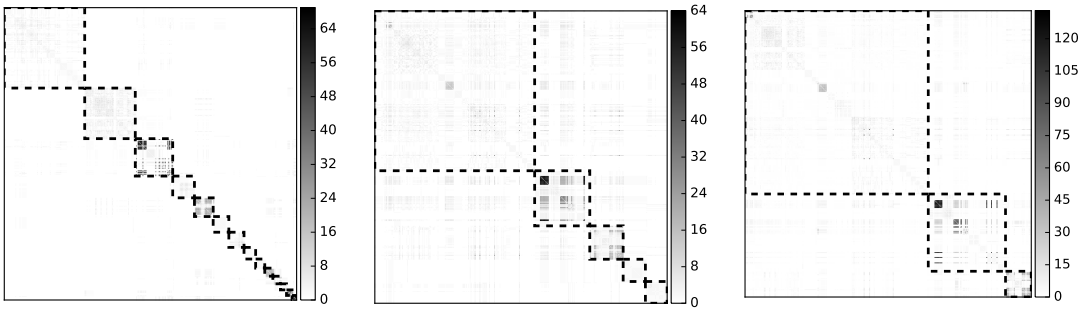
Figures 6.13 and 6.14 are results of the skeletonization with $w = 45$ and $\mathrm{P} = 60$ and 120. First of all, compared with synthetic cases or some real datasets used above, trajectory data are much more sparse. That is, most symbols in $\mathcal{B}$ appear only once in our discretized sequential databases. In results, similar results are obtained compared with Last.fm datasets, but it was difficult to find clusters automatically without tuning algorithms. In fact, only small clusters (illustrated in

††See also http://research.microsoft.com/en-us/projects/tdrive/



(a) DT (b) DP (c) DTP

Figure 6.10.: Heatmaps of User 672 from Last.fm datasets with varying DT, DP, and DTP where P $=$ 24 and $w = 2$.



(a) DT (b) DP (c) DTP

Figure 6.11.: Heatmaps of User 808 from Last.fm datasets with varying DT, DP, and DTP where P $=$ 24 and $w = 2$.

the right-bottom part) can be found: From Figure 6.13a to Figure 6.13b and Figure 6.13c, a small cluster containing a few symbols was found. By contrast, from Figure 6.14a to Figure 6.14b or Figure 6.14c, a (relatively) large cluster disappeared, and small cluster were found again.

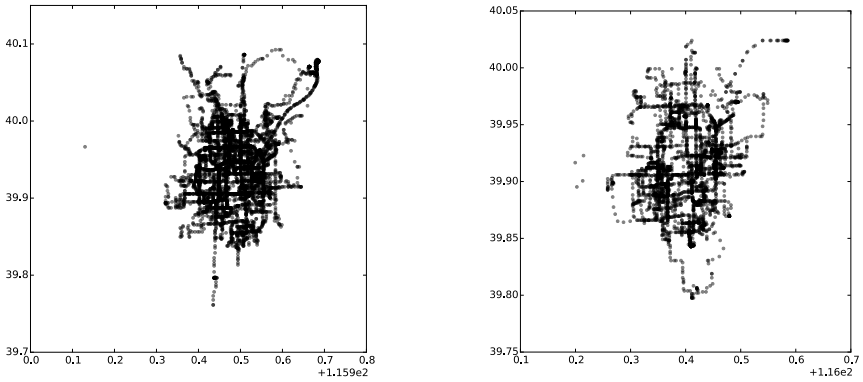
6.4.6. Discussions

Discussing the quality of clusters is fundamentally impossible as we do not have any labels. Conceptually, the skeletonization does *not* use any semantic information of symbols, and results only depend on the co-occurrences of symbols. In our method, we intend that adding more computations by the periodicity have increased information we can use in the pre-processing step. Introducing additional resources for computing the similarities such as background knowledge or taxonomy is one of interesting future work. However, such knowledge resources are in general *high cost* compared with the skeletonization. Therefore, we guess that combining both methods is much effective for solving the sparseness problem.

As another direction from the viewpoint of the increase of syntactic information when constructing similarity graphs, taking into account the *order* of indices or the *distance* of them should be an interesting future problem. These are also related to

Table 6.4.: Statistics of discretized trajectories.

ID	$\lvert\mathcal{D}\rvert$	# of non-empty (X, Y) slots ($\lvert\mathcal{B}\rvert$)
1277	8187	3410
6275	3960	2450



(a) ID 1277 (b) ID 6275

Figure 6.12.: An example of trajectory plots.

the application step of graph clustering methods. In addition, we also expect that introducing sophisticated clustering algorithms is important: For example, hierarchical spectral clustering[AS12], Non-negative Matrix Factorization (NMF) (e.g.,[CZA08]) should be helpful (e.g., considering multiple periods with hierarchy).

6.5. Mining Using Skeletonization

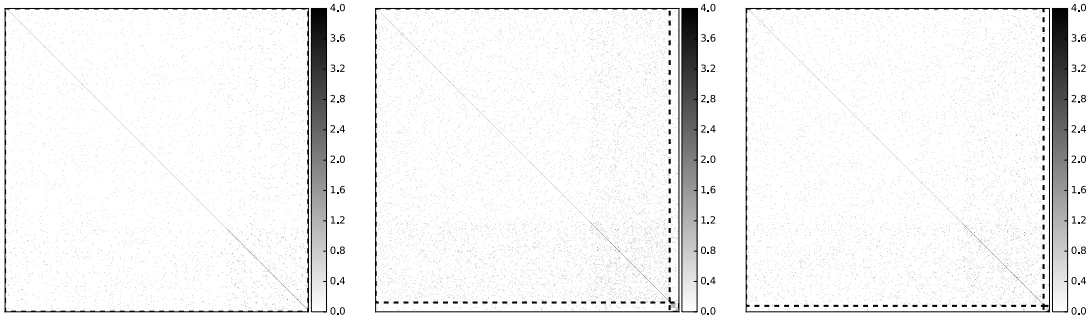
We now try to apply pattern mining algorithms based on the clusters discovered by the skeletonization techniques. In the following, we consider two cases: 1) We have some assumption on P, and 2) we have no idea on P. For the case 1), we solve the FPPPM problem with a fixed P we have in mind. For the case 2), we allow for patterns contain a gap among symbols in the following form based on[ZKCY07]: Let $p = c_1 c_2$, where $c_1, c_2 \in \mathcal{B}$ and $[\alpha, \beta]$ be an closed interval of integers. Then, we interpret the pattern p as follows: p has at least α and at most β wildcard characters between c_1 and c_2. For example $s_1 = abc$ and $s_2 = abbbc$, a pattern ac with $[\alpha, \beta] = [0, 2]$ *matches* with s_1, but does not match with s_2 because three b occurs between the first a and c.

Periodi Pattern with Gap

Experiments We now apply the periodic skeletonization with mining problems (e.g., the FPPPM problem). Our purpose here is to obtain readable and high-level patterns in mining by reducing the size $\lvert\mathcal{B}\rvert$. We use clustering results obtained by the above experiments.

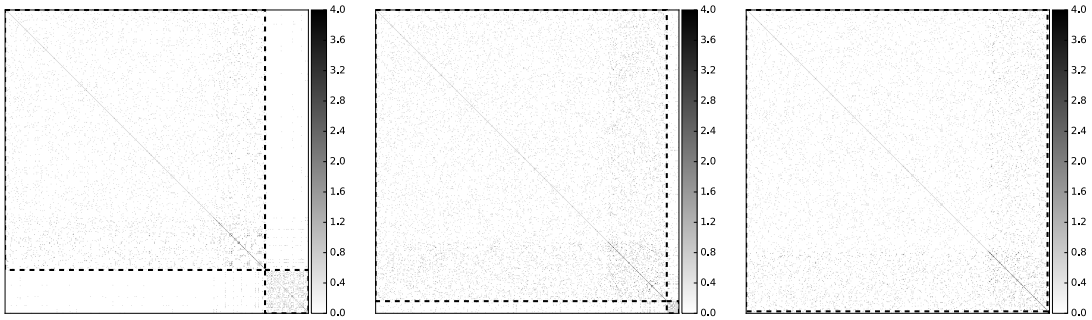
For enumeration of patterns in the case 1), we use the algorithm proposed by Yang *et al.*[YHCL13], and call it PPPMiner. For the purpose, we re-implemented PPP-

PPPMiner



(a) DT ($w = 45$) (b) DTP (P $= 60$) (c) DTP (P $= 120$)

Figure 6.13.: Heatmaps of ID 1277 with DT and DTP ($w = 45$, P $= 60$ and 120), trying with the number $k = 2$ of clusters.



(a) DT ($w = 45$) (b) DTP (P $= 60$) (c) DTP (P $= 120$)

Figure 6.14.: Heatmaps of ID 6275 with DT and DTP ($w = 45$, P $= 60$ and 120), trying with the number $k = 2$ of clusters.

MINER in Python 2.7.8. Experimental settings are the same to those in Section 6.4.2. To examine how the periodic skeletonization affects enumerating patterns by the PPPMINER, we use Kyoto and Last.fm datasets. On the other hand, for the case 2) we adopt the algorithm by Zhang et al[ZKCY07], which is also re-implemented, named by GAPMINER, in Python 2.7.8.

GAPMiner

Based on the number k of clusters, we propose an *incremental method*, where users replace symbols with cluster labels incrementally. The overview of this process is shown in Algorithm 6. We first sort clusters by the size (in Line 5). In descending order of the size, we incrementally re-encode an original sequence s with cluster labels. That is, we first replace symbols in s belonging to the largest cluster label C_1 (this new sequence is denoted by $s^{(\geq 1)}$ in Algorithm 6). We then do the same with the second largest cluster C_2, and continue this replacement (corresponding to Line 6 and Line 7). For each step in Line 8, we apply a mining method.

6.5.1. Mining with the PPPMiner

In the following experiments, we adopt both (temporal and periodical) information in the clustering step. For the Kyoto dataset, using $k = 3$ clusters, we prepared four cases: Kyoto (original), Kyoto$^{(\geq 1)}$, Kyoto$^{(\geq 2)}$, and Kyoto$^{(\geq 3)}$ to apply the PPPMINER.

Algorithm 6 Incremental Mining with Skeletonization

Input: A input sequence $\mathcal{S}$ and a parameter θ

Call INC-MS ($\mathcal{S}, \theta$) until the output $\mathcal{F}$ of mining is enough for the purpose.

1: **procedure** INC-MS ($\mathcal{S}$, θ)
2: Construct a similarity graph G from s
3: Estimate the number k of clusters (with heuristics)
4: Compute assignments of symbols ▷ $\mathcal{B} \to \{1, 2, \ldots, k\}$
5: Sort clusters $C_1, \cdots C_k$
with their cardinality ($|C_1| > |C_2| > \ldots |C_k|$, descending)
6: **for** $j = 1$ to k **do** ▷ (Or, any j $(1 \leq j \leq k)$ if you want)
7: Replace symbols in s using the cluster C_j
and get the encoded sequence $s^{(\geq j)}$ ▷ Re-encoding
8: Apply a mining method to $s^{(\geq j)}$ with θ

Table 6.5.: Numbers of enumerated patterns with (i.e., re-encoding labeled as $^{(\geq j)}$) and without the skeletonization with the PPPMiner

(a) For the Kyoto dataset (P $= 365$)

Datasets	θ=0.9	0.7	0.5	$\lvert\mathcal{B}\rvert$
Kyoto	0	0	0	359
Kyoto$^{(\geq 1)}$	9,065	57,596	133,027	224
Kyoto$^{(\geq 2)}$	28,134	210,806	523,021	97
Kyoto$^{(\geq 3)}$	54,354	349,648	917,403	3

(b) For User 672 dataset (P $= 24$)

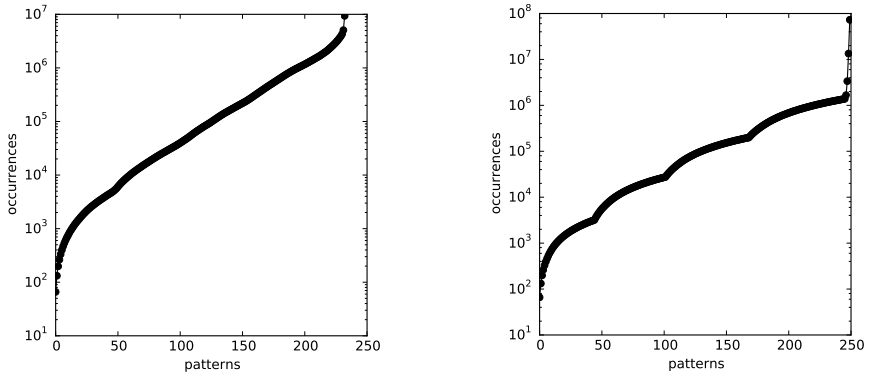
Datasets	θ =0.3	0.2	0.1	$\lvert\mathcal{B}\rvert$
User 672	0	0	0	247
User 672$^{(\geq 1)}$	128	318	51,304	177
User 672$^{(\geq 3)}$	128	319	22,540	144
User 672$^{(\geq 10)}$	127	260	5,718	10

We show the number of enumerated patterns with P $= 365$ and with varying θ in Table 6.6a. For the User 672 dataset from the Last.fm dataset, we use the number $k = 10$ of clusters to pre-process. Out of $k = 10$ clusters illustrated in Figure 6.10c, for the integer j in Line 6, we use the largest cluster C_1 and get the re-encoded sequence User 672$^{(\geq 1)}$ corresponding to $j = 1$. In the same manner, we adopt the top three largest clusters $C_1, C_2,$ and C_3 (i.e., $j = 3$) and get the sequence User 672$^{(\geq 3)}$. We finally use all clusters ($j = 10$) and label the obtained sequence as User 672$^{(\geq 10)}$. We show in Table 6.6b the numbers of enumerated patterns.

Discussions In both cases we cannot find any frequent patterns *without* the periodic skeletonization. That is, without any pre-processing, databases are sparse and we cannot evaluate the support well to get insights from datasets in the form of partially periodic patterns. However, with thanks to the periodic skeletonization, we can find many frequent patterns in other cases.

Table 6.7.: Numbers of enumerated patterns for ID 1277 trajectory, with threshold 0.00022 of the gap $[\alpha, \beta] = [0, 5]$ by using the GAPMiner.

Datasets	Length 3	4	5	6
ID 1277$^{(\geq 1)}$	46	58	68	78
ID 1277	72	59	51	51



(a) From ID 1277 (b) From ID 1277$^{(\geq 1)}$

Figure 6.15.: Two logarithmic plots of the support of patterns enumerated by the GAPMiner from two sequences.

Because the periodic skeletonization help us to find rough, characteristic patterns by clustering, we can find abstract but readable and high-level frequent patterns. For example in the Kyoto$^{(\geq 1)}$ setting, we can find 9,065 patterns which characterize 90% of segments in the given sequence. In addition, in the settings of User 672$^{(\geq 1)}$ and User 672$^{(\geq 3)}$, we *successfully* find roughly 300 frequent patterns that characterize 20% of segments, and this number of patterns is relatively small and easy to analyze.

We confirmed that clustering using the skeletonization as a pre-processing of pattern mining work well to get more frequent patterns than those obtained from raw sequences.

6.5.2. Mining with the GAPMiner

We adopt the trajectory sequence of ID 1277 used the above. In this case, we use the number $k = 2$ of clusters because we have no idea on the numbers of discretized trajectories. Then the largest cluster C_1 is replaced and a new encoded dataset, denoted by ID 1277$^{(\geq 1)}$, is obtained. We let $[\alpha, \beta]$ – the gap admitted among characters – to be $\alpha = 0$ and $\beta = 5$, and mine patterns with gaps up to the length 6 as an example. As shown in Table 6.7, we could find almost similar numbers of patterns of length from 3 to 6.

Discussions To investigate the differences of enumerated patterns by using the support of patterns we give logarithmic plots of them in Figure 6.15. The figure shows the difference of patterns enumerated by GAPMiner clearly. Without apply-

ing any pre-processing methods to databases, in Figure 6.15a, the support varies in exponential. On the other hand, by the skeletonization in Figure 6.15b, patterns can have similar supports, as results of removing rare symbols from $\mathcal{B}$ by replacing the symbols in the large cluster discovered with a cluster label. This result appeared as an waved curve on the logarithmic plot in Figure 6.15b.

Through experiments, we confirmed that the skeletonization affected the distribution of the frequency of patterns. We have many choices on how to replace symbols originally in $\mathcal{B}$ with cluster labels, and in these experiments we use cluster labels in an descending order.

6.5.3. Future Directions

It is always an open problem in pattern mining how to deal with a kind of redundancy among patterns. In our experiments, many patterns constructed by shifting symbols are mined. For example, we often have the case a frequent pattern p occurring at 8 a.m. again becomes frequent patterns at 9 a.m., 10 a.m., and so on. That is, the primitive definition of partially periodic patterns have some redundancy. It is our important future work to overcome the redundancy problem by considering well-studied conception (e.g., *closed* or *maximal* patterns) and combining them with our pre-processing method. Tuning hyper-parameters including the number k of clusters and the width r of sliding windows is also our future work to enrich partially periodic pattern mining with pre-processing using the skeletonization.

In addition, further studies are possible in pre-processing of the KDD process beyond the last experiments using the plots of frequencies of patterns. Figure 6.16 is a logarithmic plot (in the y-axis) representing the frequency of each symbol[♯♯] in $\mathcal{B}$ for the ID 1277 sequence. If given databases are not merely noise, we have an assumption that the symbol frequency is not uniform. This fact encourages more developments on applications of Machine Learning methods for the sparseness of several pattern mining problems. For example considering the TF-IDF model in pattern mining, utility-based pattern mining, and unsupervised metric learning on symbols area related topics from the viewpoint of pre-processing using machine learning algorithms.

6.6. Summary

In this chapter we provide a new skeletonization method for dealing with partially periodic patterns based on the temporal skeletonization and periodic information. Our experiments with synthetic and real datasets show that our method could help us to obtain clusters of symbols even for periodic settings, particularly for sequences having only one fixed period. Pattern mining results with the skeletonization indicate that our method is helpful to obtain readable results with a relatively small computational cost as $\mathcal{B}$ get small. Even we use a large threshold,

[♯♯] It is equal to the support of each symbol, i.e., the total number of occurrences of symbols in a database.

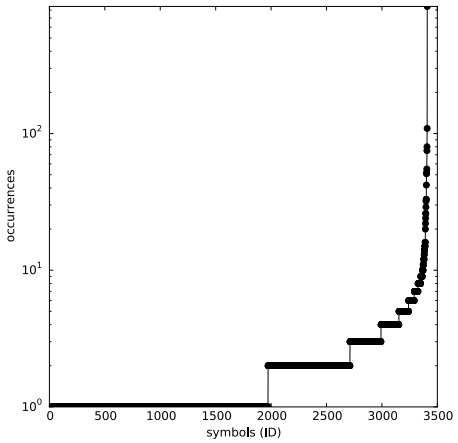


Figure 6.16.: The logarithmic plot of the frequency of symbols.

we can find frequent patterns which cannot be listed without the skeletonization. Furthermore, in unstable cases (of the period of sequences), it is clarified that our skeletonization work well to learn hidden clusters from sequences. Using more real datasets, we test that our method give some insights on relation of symbols used for describing databases, and their analyses might be important and helpful for Knowledge Discovery.

We would like to develop algorithms to reduce the redundancy of patterns more, based on well-studied concepts (e.g., closed patterns) together with the skeletonization. Further discussion using other pre-processing methods, particularly comparing methods using semantic information (i.e., hierarchy of symbols, background knowledge) are also our important future work.

CHAPTER 7

Conclusion

This thesis focused on the three main parts in a common model of Data Mining, the KDD process[FPsS96], consisting of 1) pre-processing of data, 2) pattern mining from the refined data, and 3) post-processing of the output of pattern mining. Based on preliminaries of pattern mining, shortly summarized in Chapter 2, each section treated one of the problems along with the KDD process. Pre-processing for effective pattern mining was discussed in Chapter 6. Theoretical analyses of pattern mining was given in Chapter 3. Post-processing of pattern mining using lattices based on algorithmic approaches was investigated in Chapter 4 and Chapter 5. Objects throughout of the thesis are graphs given or generated by various manners in Data Mining from and using graphs.

The results indicated the importance of the developments of methods for enhancing the utility, the usability, and the effectiveness of pattern mining in Knowledge Discovery via graphs. In addition, the results suggested that using implicit assumptions in the KDD process to design algorithms are effective. For example, in Chapter 3, common properties of given databases were used to design and investigate mining algorithms. In Chapter 4, naturally available side information was embedded into operators to compare episodes, a class of directed acyclic graphs. In Chapter 6, definitions of patterns are adopted to Machine Learning-based pre-processing to clarify given data from the viewpoint of data cleaning.

In summary, the core strategy throughout the thesis is to develop methods for various problems in the KDD process for graphs based on our assumption that Knowledge Discovery can be achieved by solving not only one-time-only problems, but also iterative (multiple) problems simultaneously and continuously. We have construct methods in the thesis to take into account such iterative applications of algorithms to databases, for enhancing the utility and the effectiveness of pattern mining in Knowledge Discovery.

The followings are lists of future perspectives of pattern mining research and problem settings beyond the thesis.

1. Supervised and unsupervised pattern mining: When no labels are given (i.e., in *unsupervised* mining problems mainly discussed in the thesis), evaluations of methods at pre-processing and post-processing are challenging compared with the pattern mining part in which computational complexity (time complexity, space complexity, listing complexity, etc.) is adopted. To discuss methods with labels, adopting datasets with labels is a fundamental approach (e.g.,[LOP15], Section **??**). It

is also known as *discriminative pattern mining*, which is a famous variation in the literature with statistical evaluations or testing (e.g., [MS00,NBT07,CYHH07,NLW09]).

If we have labels of data, we can directly apply the statistical hypothesis testing (e.g., Fisher's exact test) to evaluate patterns in the sense of some statistical test. As the number of patterns is huge because of the pattern combinatorial explosion, recent researches have been focused on multiple statistical test (e.g., recent papers [TOHTS13,TTS13,MUT+14,SLKB15,LSPB15,TKS15]).

In addition, considering hypothesis could be possible by testing whether or not the given database $\mathcal{D}$ obeys a hypothesis $\mathcal{H}$ without explicitly describing models (e.g., code-tables are explicitly defined models in Chapter 5). The method named *swap randomization* (e.g., [GMMT07,HOV+09,KDB10]) has been investigated as a tool of testing such hypotheses, particularly for itemsets in that itemsets can easily modeled by probabilistic distributions. Generalizing such strategies for structured data is an important future research direction.

2. Quantification of behavior of iterative pattern mining: Judging from experimental results in the thesis, we strongly believe that our work focusing on the pre-processing and the mining task could be important. We conjecture that such methods can be also discussed (without labels) based on the idea of the grid search strategy of machine learning in hyperparameter optimization in that we vary the parameter θ from $50\%, 40\%, 30\%, 20\%, 10\%, \ldots$ in mining. We partially discussed this problem in Section 6.5 by checking the spectrum of frequent patterns enumerated. The studies related to spectrums have been attracted much attention recently (e.g., [BP14,vLU14]). Using spectrum of patterns, we could formulate a behavior of some algorithm with a database $\mathcal{D}$ and a parameter θ.

How changes of parameters affect pattern mining results could be also related to the *sensitivity* of a function, which is often used in the research area of pattern mining under privacy preserving constraints: Consider a function $f(\cdot)$ getting a database $\mathcal{D}$ and returning some real value. The *sensitivity* of the function f with respect to the database is defined as

$$\Delta f = \max_{\mathcal{D}_1, \mathcal{D}_2} ||f(\mathcal{D}_1) - f(\mathcal{D}_2)||_1,$$

where $||\cdot||_1$ represents some norm and max is checked for all database pairs $(\mathcal{D}_1, \mathcal{D}_2)$ which are in the *neighbor* relation (i.e., $\mathcal{D}_1$ and $\mathcal{D}_2$ are differing in at most one transaction). Though this measure is not directly applicable to listing algorithms, the methods related to evaluate a function itself could be regarded as a start point to build new formulations of evaluations of listing algorithms, beyond their listing complexity (studied in Chapter 3). This perspective arises from the fact that listing algorithms can be regarded as a kind of function to generate some distribution on a set $\mathcal{F}$ of frequent patterns.

3. Issues on probability/uncertainty and utility: Weights of items (e.g., prices, priority in a wishlist, etc.) have been a formulation to take into account the importance of elements in the base-set $\mathcal{B}$ to reflect users' interests or preferences (e.g., [HM07,EGA08]). Values of the weights are often referred as *utilities*. Roughly speak-

ing, miming patterns by taking into account utilities of patterns and elements in $\mathcal{B}$ is more difficult than mining ordinal patterns. Particularly, keeping the anti-monotonicity of a measure is a crucial problem for efficient enumeration.

Independently of utility pattern mining, another famous variation is *uncertainty* of transactions as *uncertain databases* (e.g.,[CKH07,AY09,Agg09]). Methodologies for uncertain data are related to Chapter 4 to prune the space of pattern concepts in our study, and to provide a good measure to evaluate enumerated concepts without labels. As utilities can be normalized, methods for utility pattern mining and those for uncertain pattern mining could be adopted simultaneously to achieve effective pattern mining. From viewpoints of model selections (related to Chapter 5), characterizations of mining algorithms and patterns (related to Section 6.5 as we mentioned above) in probabilities and uncertainty could be valuable, and it could be a future research topic to investigate these approaches for complex data (beyond binary data in FCA or itemsets).

From the viewpoint of probabilistic models, sampling or approximating the set $\mathcal{F}$ directly under a given constraint described as a probabilistic distribution $\Pr(\cdot)$ is an alternative approach for pattern set mining (e.g., approximation[AGM04], sampling methods[BLPG11,MBG14]). Therefore, combining such models with algorithmic approaches for unsupervised pattern set mining could be important research topic.

APPENDIX A

Symbols and Mathematical Background

Mathematical backgrounds are listed, particularly for lattices (used in Chapters 4 and 5).

Relation, Order, and Lattice

Binary Relation

Definition A.1 (Binary Relation) A binary relation R between two finite sets E and F defined on the product $E \times F$ consisting of pairs (r_e, r_f) with $r_e \in E$ and $r_f \in F$.

Partial Order Relation

Definition A.2 (Partial Order Relation) A *binary relation* R on a finite set E is called an *partial order relation* if it satisfies the following conditions for all elements $x, y, z \in E$: 1) Reflexivity; $(x, x) \in R$, 2) Anti-symmetry; $(x, y) \in R$ and $x \neq y$ implies $(y, x) \notin R$, and 3) Transitivity; $(x, y) \in R$ and $(y, z) \in R$ implies $(x, z) \in R$.

Partially Ordered Set

Definition A.3 (Partially Ordered Set) Given a partial order $\preceq$ on a finite set E, a *poset* (i.e., *partially ordered set*) is a pair $(E, \preceq)$.

If it holds either $a \leq b$ or $b \leq a$ for any $a, b \in E$, the partial order is a *total order*. For example, the set of all itemset $2^{\mathcal{B}}$ is a poset with the subset relation $\subseteq$.

Infimum
Supremum

Definition A.4 (Infimum, Supremum) Let $(E, \preceq)$ be a poset and $A \subseteq E$. A *lower bound* of A is an element $s \in E$ with $s \preceq a$ for all $a \in A$. If it exists a largest element in the set of all lower bounds of A, it is called the *infimum* of A, denoted by $\inf A$. Dually, a *upper bound* is defined, and a least upper bound is called *supremum*, denoted by $\sup A$.

Sometimes the infimum and the supremum of A is referred to be *meet* and *join*, respectively.

Lattice

Definition A.5 (Lattice) A poset $(E, \preceq)$ is a *lattice* if for any two elements $x, y \in E$ the supremum $x \vee y$ and the infimum $x \wedge y$ always exist. The poset is called a *complete lattice* if for any subset $X \subseteq E$, the supremum $\bigvee X$ and the infimum $\bigwedge X$ exist. Every complete lattice has a largetst element $\mathbf{1}_E$ called the *unit* element and the smallest element $\mathbf{0}_E$ called the *zero* element.

Join Semi-lattice
Meet Semi-lattice

Definition A.6 (Join Semi-lattice, Meet Semi-lattice) A poset $(E, \preceq)$ is a *join semi-lattice* if for any two elements $a, b \in E$ the supremum $x \vee y$ always exists. Dually, it is a *meet semi-lattice* if the infimum $x \wedge y$ always exists. A lattice is a poset that is both a join semi-lattice and meet semi-lattice with respect to the same partial order $\preceq$.

Let E be a set and η a mapping from the powerset 2^E into 2^E.

Definition A.7 (Closure Operator) The function η is a *closure operator* on E if for anly sets $A, B \subseteq E$, it is 1) Extensive; $A \subseteq \eta(A)$, 2) Monotone; $A \subseteq B$ implies that $\eta(A) \subseteq \eta(B)$, and 3) Idempotent; $\eta(\eta(A)) = \eta(A)$.

A subset $A \subseteq S$ is η-closed if $A = \eta(A)$. The set of all η-closed $\{A \subseteq E \mid A = \eta(A)\}$ is called a *closure system*.

Galois Connection

Definition A.8 (Galois Connection) Let $(P, \preceq)$ and $(Q, \preccurlyeq)$ be two ordered sets. A pair (μ, γ) of maps; $\mu : P \to Q$ and $\gamma : Q \to P$ is called a *Galois connection* if it holds for $p_1, p_2 \in P$ and $q_1, q_2 \in Q$: $p_1 \preceq p_2$ implies $\mu(p_1) \succcurlyeq \mu(p_2)$, $q_1 \preccurlyeq q_2$ implies $\gamma(q_1) \succcurlyeq \gamma(q_2)$, $p \preceq \gamma \circ \mu(p)$, and $q \preccurlyeq \mu \circ \gamma(q)$.

In FCA, on the context $K = (G, M, I)$, we have a Galois connection between $(2^G, \subseteq)$ and $(2^M, \subseteq)$ with letting P be G, Q be M, $\preceq$ be $\subseteq$, and $\preccurlyeq$ be $\subseteq$.

Basic Theorem

Definition A.9 (Basic Theorem) The concept lattice $\mathfrak{P}(G, M, I)$ is a *complete lattice* in which infimum and supremum are given by

$$\bigwedge_{t \in T} (A_t, B_t) = \left(\bigcap_{t \in T} A_t, \left(\bigcup_{t \in T} B_t \right)'' \right) \text{ and} \tag{A.1}$$

$$\bigvee_{t \in T} (A_t, B_t) = \left(\left(\bigcup_{t \in T} A_t \right)'', \bigcap_{t \in T} B_t, \right), \tag{A.2}$$

where $(\cdot)'$ is a short representation of a Galois connection on $K = (G, M, I)$.

Information Theory To address the interestingness of items and patterns enumerated in pattern mining, the field *Information Theory* is the traditional research discipline, in which the notion of *entropy* introduced by Shannon 1948* measuring the *information content of a given random variable*. Formally, given a discrete random variable X with domain $\mathcal{X}$ a probability distribution p, the entropy of X, denoted by $H(X)$, is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x).$$

The base of the logarithm in the paper is set to be 2, and by convenience $0 \log 0 = 0$.

*A mathematical theory of communication

For a database $\mathcal{D}$ and an itemset X, we can now define the entropy of X as

$$H(X, \mathcal{D}) = -\sum_{x \in \mathcal{X}} \text{freq}(X = x) \log \text{freq}(X = x),$$

where $\mathcal{X} = dom(X)$.

The problem of finding a *model* for a given database $\mathcal{D}$ is not trivial and an important background of Knowledge Discovery. Model

Proposition A.1 (Occam's razor) Given two models of equal explanatory power, the *simplest* model is to be preferred.

The *minimum description length* (MDL) principle is known as a practical version of *Kolmogorov Complexity*. Given a database $\mathcal{D}$ and a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ according to the MDL principle is the one that minimizes

$$L(M) + L(\mathcal{D} \mid M),$$

where

- $L(M)$ is the length in bits of the description of a model M, and
- $L(\mathcal{D} \mid M)$ is the length in bits of the description of the data $\mathcal{D}$ encoded with a given model M.

Foundations

$\{\cdots\}$	Set; If it is clear from the context, the parenthes are omitted; e.g., $\{i, j, k\}$ is represented by ijk
$\langle\cdots\rangle$	Sequence; If it is clear from the context, the parenthes are omitted; e.g., $\langle a, b, c\rangle$ is represented by abc. For an element h and a sequence H, the list $\langle h \mid H\rangle$ is obtained by inserting h before H.
$\mathbb{1}[\textbf{pred}]$	Indicator function; If the given predicate **pred** is **true**, returns 1 and 0 otherwise.
2^A	Powerset of a finite set A
$\mathbb{N}$	The set of natural numbers including 0

Sequences, Strings, and Episodes

λ	The empty string
$\mathcal{B}$	Alphabet (event symbols)
$\mathcal{B}^*$	The set of finite sequences / strings over $\mathcal{B}$
$\mathcal{B}^+$	The set of finite sequences / strings without λ
$\lvert w\rvert$	The length of w
$a \mapsto B \mapsto c$	A diamond episode $\langle a, B, c\rangle$
$A \mapsto B$	A bipartite episode $\langle A, B\rangle$ (a.k.a., a rule $A \to B$)
$\sqcap_{\mathcal{B}}$	The binary operation of comparing symbols in $\mathcal{B}$
$\bot_{\mathcal{B}}$	The infimum (bottom) of the comparison by $\sqcap_{\mathcal{B}}$
$(\mathcal{B}, \sqcap_{\mathcal{B}})$	Meet semi-lattice of symbols

Formal Concept Analysis, Pattern Structure Analysis

O	The set of objects
A	The set of attributes
I	Binary relation between O and A
$K = (O, A, I)$	(Formal) context
$(\cdot)^{\uparrow}, (\cdot)^{\downarrow}$	Galois connection for FCA
$\preceq$	Partial order between concepts
$\langle \mathfrak{L}, \preceq\rangle$	Concept lattice, where $\mathfrak{L}$ is a set of all concepts
D	The set of descriptions
$\sqcap$	Binary operation among descriptions
$(D, \sqcap)$	Meet semi-lattice
δ	Extractor; Mapping from O to D
$\mathbb{P} = (O, (D, \sqcap), \delta)$	Pattern structure
$(\cdot)^{\square}, (\cdot)^{\diamond}$	Galois connection for PSA

Information Theory and MDL

$H(X)$	Entropy on a discrete probability distribution p and a random variable X
$\mathcal{M}$	Set of models
$M, M_1, M_2, \ldots$	Models
$L(\mathcal{D})$	Description length of a database $\mathcal{D}$
$L(\mathcal{D} \mid M)$	Description length of a database $\mathcal{D}$ when the model M is given

Major Publications

Journal Articles

[J1] Otaki, K., Yamamoto, A.: Periodic Pattern Mining with Periodical Co-occurrences of Symbols, *IPSJ Transactions on Mathematical Modeling and Its Applications* (*TOM*), vol.9(1), pp.32–42, 2016.

Peer-reviewed Conference Proceedings

[P1] Otaki, K., Yamamoto, A.: Periodical Skeletonization for Partially Periodic Pattern Mining, *Discovery Science* 2015 (LNCS 9356), pp.186-200, 2015 (long paper).

[P2] Otaki, K., Yamamoto, A.: Edit Operations on Lattices for MDL-based Pattern Summarization, *FCA&A@ICFCA2015* (Electric Online Proceedings), CEUR Workshop Proceedings Vol.1434, pp.17-32, 2015.

[P3] Otaki, K., Yamamoto, A.: Pattern Structures for Understanding Episode Patterns, *Concept Lattice and Their Applications* (Electric Online Proceedings), CEUR Workshop Proceedings Vol.1252, pp.47-58, 2014.

[P4] Horváth, T., Otaki, K., Ramon, J. (alphabetical order): Efficient Frequent Connected Induced Subgraph Mining in Graphs of Bounded Tree-Width, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD* 2013 (LNCS 8188), pp.622-637, 2013.

All Other Research Activities by the author

Journal Activities

[J2] Otaki, K., Sugiyama, M., Yamamoto, A.: Privacy Preserving Using Dummy Data for Set Operations in Itemset Mining Implemented with ZDDs, *IEICE Transactions on Information and Systems*, vol.95-D(12), pp.3017-3025, 2012.

[J3] Ouchi, S., Okayama, T., Otaki, K., Yoshinaka, R., Yamamoto, A.: Learning Concepts and Their Unions from Positive Data with Refinement Operators, *Annals of Mathematics and Artificial Intelligence*, to appear.

Peer-reviewed Conference Proceedings

[P5] Okayama, T., Yoshinaka, R., Otaki, K., Yamamoto, A.: A Sufficient Condition for Learning Unbounded Unions of Languages with Refinement Operators, *International Symposium on Artificial Intelligence and Mathematics* (Electric Online Proceedings), 2014.

[P6] Sugiyama, M., Otaki, K.: Detecting Anomalous Subgraphs on Attributed Graphs via Parametric Flow, *New Frontiers in Artificial Intelligence* (JSAI-isAI Workshops 2014, LNCS 9067), pp.340-355, 2014.

[P7] Ikeda, M., Otaki, K., Yamamoto, A.: Formal Concept Analysis for Process Enhancement on a Pair of Perspectives, *Concept Lattice and Their Applications* (Electric Online Proceedings), CEUR Workshop Proceedings Vol.1252, pp.59-70, 2014.

[P8] Kondo, S., Otaki, K., Ikeda, M., Yamamoto, A.: Fast Computation of the Tree Edit Distance between Unordered Trees Using IP Solvers, *Discovery Science* 2014 (LNCS 8777), pp.156-167, 2014 (long paper).

Peer-reviewed Talks

[T1] Otaki, K., Yamamoto, A.: Probabilistic Models Based on Regular Pattern Languages and Their Learning Problems, Work-In-Progress session in 12th International Conference on Grammatical Inference, 2014.

Domestic Conferences

[D1] Otaki, K., Yamamoto, A.: Mining Periodic Patterns of Sequences via Skeletonization, IPSJ Annual Meetings (IPSJ 2015), Vol.1, pp.447-448, 2015.

[D2] Otaki, K., Yamamoto, A.: Periodic Pattern Mining with Periodical Co-occurrences of Symbols, IPSJ SIG Technical Report, Vol.2015-MPS-105, np.7, 2015, **Best Presentation Award**.

References

[AF96] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1–3):21 – 46, 1996. First International Colloquium on Graphs and Optimization.

[Agg09] C. C. Aggarwal. *Managing and Mining Uncertain Data*, volume 35 of *Advances in Database Systems*. Kluwer, 2009.

[AGM04] F. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *Proceedings of the 10th KDD*, pages 12–19, 2004.

[AH14] C. C. Aggarwal and J. Han, editors. *Frequent pattern mining*. Springer International Publishing, 2014.

[AIS93] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Record*, 22(2):207–216, 1993.

[AMS+96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. 1996.

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11 – 24, 1989.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th VLDB*, pages 487–499, 1994.

[AS95] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th ICDE*, pages 3–14, 1995.

[AS12] C. Alzate and J. A. Suykens. Hierarchical kernel spectral clustering. *Neural Networks*, 35(0):21 – 30, 2012.

[AY09] C. C. Aggarwal and P. S. Yu. A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):609–623, 2009.

[BAG00] R. J. Bayardo, Jr., R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. *Data mining and knowledge discovery*, 4(2-3):217–240, 2000.

[Bay98] R. J. Bayardo, Jr. Efficiently mining long patterns from databases. *SIGMOD Record*, 27(2):85–93, 1998.

[BCG01] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th ICDE*, pages 443–452, 2001.

[BLPG11] M. Boley, C. Lucchese, D. Paurat, and T. Gärtner. Direct local pattern sampling by efficient two-step random procedures. In *Proceedings of the 17th KDD*, pages 582–590, 2011.

[Bod90] H. L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.

[Bod96] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[Bod98] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, 1998.

[BP14] C. Borgelt and D. Picado-Muiño. Simple pattern spectrum estimation for fast pattern filtering with CoCoNAD. In *Proceedings of the 13th IDA*, pages 37–48, 2014.

[CDK+99] S. Chakrabarti, B. E. Dom, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J. Kleinberg. Mining the web's link structure. *Computer*, 32(8):60–67, 1999.

[Cel10] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[CG02] T. Calders and B. Goethals. Mining all non-derivable frequent itemsets. In *Proceedings of the 6th PKDD*, pages 74–86. Springer, 2002.

[CH08] G. Cormode and M. Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[CKH07] C.-K. Chui, B. Kao, and E. Hung. Mining frequent itemsets from uncertain data. In *Proceedings of the 11th PAKDD*, pages 47–58, 2007.

[CNFF96] D. W. Cheung, V. T. Ng, A. W. Fu, and Y. Fu. Efficient mining of association rules in distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):911–922, 1996.

[CYHH07] H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *Proceedings of the 23rd ICDE*, pages 716–725, 2007.

[CZA08] A. Cichocki, R. Zdunek, and S.-I. Amari. Nonnegative matrix and tensor factorization [lecture notes]. *Signal Processing Magazine, IEEE*, 25(1):142–145, 2008.

[DKWK05] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.

[DL99] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the 5th KDD*, pages 43–52, 1999.

[DRZ07] L. De Raedt and A. Zimmermann. Constraint-based pattern set mining. In *Proceedings of the 7th SDM*, pages 237–248, 2007.

[DTK98] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 30–36, 1998.

[EGA08] A. Erwin, R. P. Gopalan, and N. Achuthan. Efficient mining of high utility itemsets from large datasets. In *Proceedings of the 12th PAKDD*, pages 554–561, 2008.

[For10] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[FPsS96] U. Fayyad, G. Piatetsky-shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[Fri97] J. H. Friedman. Data mining and statistics: What's the connection? In *Proceedings of the 29th Symposium on the Interface Between Computer Science and Statistics*, 1997.

[Gä03] T. Gärtner. A survey of kernels for structured data. *SIGKDD Explorations Newsletter*, 5(1):49–58, 2003.

[GD05] L. Getoor and C. P. Diehl. Link mining: A survey. *SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

[GFW03] T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th COLT / 7th Kernel (COLT/Kernel 2003)*, pages 129–143, 2003.

[GGM04] F. Geerts, B. Goethals, and T. Mielikäinen. Tiling databases. In *Proceedings of the 7th DS*, pages 278–289, 2004.

[GHP+03] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212, 2003.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guid to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GLH15] S. García, J. Luengo, and F. Herrera. *Data preprocessing in data mining*. Springer, 2015.

[GMMT07] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data*, 1(3):14, 2007.

[GMS04] A. Gionis, H. Mannila, and J. Seppänen. Geometric and combinatorial tiles in 0—1 data. In *Proceedings of the 8th PKDD*, pages 173–184, 2004.

[GRS99] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: sequential pattern mining with regular expression constraints. In *Proceedings of the 25th VLDB*, pages 223–234, 1999.

[GZ03] B. Goethals and M. J. Zaki, editors. *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[GZ05] K. Gouda and M. J. Zaki. Genmax: An efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery*, 11(3):223–242, 2005.

[HCXY07] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[HD09] J. Han and B. Ding. Stream mining. In *Encyclopedia of Database Systems*, pages 2831–2834. 2009.

[HDY99] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of 15th ICDE*, pages 106–115, 1999.

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st VLDB*, pages 420–431, 1995.

[HGY98] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *Proceedings of the 4th KDD*, pages 214–218, 1998.

[HLN99] J. Han, L. V. Lakshmanan, and R. T. Ng. Constraint-based, multidimensional data mining. *Computer*, 32(8):46–50, 1999.

[HLXC12] P. Huang, C.-J. Liu, L. Xiao, and J. Chen. Wireless spectrum occupancy prediction based on partial periodic pattern mining. In *Proceedings of the 20th MASCOTS*, pages 51–58, 2012.

[HM07] J. Hu and A. Mojsilovic. High-utility pattern mining: A method for discovery of high-utility item sets. *Pattern Recognition*, 40(11):3317–3324, 2007.

[HN07] M. Hajiaghayi and N. Nishimura. Subgraph isomorphism, log-bounded fragmentation, and graphs of (locally) bounded treewidth. *Journal of Computer and System Sciences*, 73(5):755 – 768, 2007.

[HOV+09] S. Hanhijärvi, M. Ojala, N. Vuokko, K. Puolamäki, N. Tatti, and H. Mannila. Tell me something i don't know: randomization strategies for iterative data mining. In *Proceedings of the 15th KDD*, pages 379–388, 2009.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the SIGMOD2000*, pages 1–12, 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.

[HR10] T. Horváth and J. Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theoretical Computer Science*, 411(31–33):2784 – 2797, 2010.

[HWLT02] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *Proceedings of the 2002 ICDM*, pages 211–218, 2002.

[HYH+05] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl 1):i213–i221, 2005.

[IWM00] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th PKDD*, pages 13–23, 2000.

[JGZ04] R. J. B. Jr., B. Goethals, and M. J. Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[JS02] S. Jaroszewicz and D. A. Simovici. Pruning redundant association rules using maximum entropy principle. In *Proceedings of the 6th PAKDD*, pages 135–147. 2002.

[JYP88] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119 – 123, 1988.

[KAH09] T. Katoh, H. Arimura, and K. Hirata. Mining frequent bipartite episode from event sequences. In *Proceedings of the 12th DS*, pages 136–151, 2009.

[Kat11] T. Katoh. *Efficient Algorithms for Extracting Frequent Episodes from Event Sequences*. PhD thesis, Hokkaido University, 2011.

[KDB10] K.-N. Kontonasios and T. De Bie. An information-theoretic approach to finding informative noisy tiles in binary databases. In *Proceedings of the 18th SDM*, pages 153–164, 2010.

[KK01] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 ICDM*, pages 313–320, 2001.

[KMM04] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, pages 729–736, 2004.

[KTI03] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th ICML*, pages 321–328, 2003.

[LG13] A. Le and M. Gertz. Mining periodic event patterns from rdf datasets. In B. Catania, G. Guerrini, and J. Pokorný, editors, *Proceedings of the 17th ADBIS*, pages 162–175, 2013.

[LKFT04] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski. Subgroup discovery with CN2-SD. *The Journal of Machine Learning Research*, 5:153–188, 2004.

[LOP06] C. Lucchese, S. Orlando, and R. Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):21–36, 2006.

[LOP15] C. Lucchese, S. Orlando, and R. Perego. Supervised evaluation of top-k itemset mining algorithms. In *Proceedings of the 17th DaWaK*, pages 82–94, 2015.

[LSPB15] F. Llinares-López, M. Sugiyama, L. Papaxanthos, and K. M. Borgwardt. Fast and memory-efficient significant pattern mining via permutation testing. In *Proceedings of the 21st KDD*, pages 725–734, 2015.

[LZX+14] C. Liu, K. Zhang, H. Xiong, G. Jiang, and Q. Yang. Temporal skeletonization on sequential data: Patterns, categorization, and visualization. In *Proceedings of 20th KDD*, pages 1336–1345, 2014.

[Mat78] D. W. Matula. Subtree isomorphism in $o(n^{\frac{5}{2}})$. *Annals of Discrete Mathematics*, 2:91–106, 1978.

[MBG14] S. Moens, M. Boley, and B. Goethals. Providing concise database covers instantly by recursive tile sampling. In *Proceedings of the 17th DS*, pages 216–227, 2014.

[MR13] C. H. Mooney and J. F. Roddick. Sequential pattern mining – approaches and algorithms. *ACM Computer Surveys*, 45(2):19:1–19:39, 2013.

[MS00] S. Morishita and J. Sese. Transversing itemset lattices with statistical metric pruning. In *Proceedings of the 19th PODS*, pages 226–236, 2000.

[MSU+01] T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda. Discovery of frequent tree structured patterns in semistructured web documents. In *Proceedings of the 5th PAKDD*, pages 47–52, 2001.

[MT92] J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1–3):343 – 364, 1992.

[MT97] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[MTIV97] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[MUT+14] S.-i. Minato, T. Uno, K. Tsuda, A. Terada, and J. Sese. A fast method of statistical assessment for combinatorial hypotheses based on frequent itemset enumeration. In *Proceedings of the ECML PKDD 2014*, pages 422–436, 2014.

[NBT07] S. Nowozin, G. Bakir, and K. Tsuda. Discriminative subsequence mining for action classification. In *Proceedings of the 11th ICCV*, pages 1–8, 2007.

[NCW97] S.-H. Nienhuys-Cheng and R. d. Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[New04] M. E. Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

[New06] M. E. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[NJW01] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 13 (NIPS 2001)*, pages 849–856, 2001.

[NLHP98] R. T. Ng, L. V. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *ACM SIGMOD Record*, volume 27, pages 13–24, 1998.

[NLW09] P. K. Novak, N. Lavrač, and G. I. Webb. Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *The Journal of Machine Learning Research*, 10:377–403, 2009.

[PBTL99] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th ICDT*, pages 398–416, 1999.

[PCT+03] F. Pan, G. Cong, A. K. Tung, J. Yang, and M. J. Zaki. Carpenter: Finding closed patterns in long biological datasets. In *Proceedings of the 9th KDD*, pages 637–642, 2003.

[PHM00] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the DMKD2000*, pages 21–30, 2000.

[PHMa+01] J. Pei, J. Han, B. Mortazavi-asl, H. Pinto, Q. Chen, U. Dayal, and M. chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of 17th ICDE*, pages 215–224, 2001.

[PHMA+04] J. Pei, J. Han, B. Mortazavi-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(11):1424–1440, 2004.

[PTU10] S. Parthasarathy, S. Tatikonda, and D. Ucar. A survey of graph mining techniques for biological datasets. In C. C. Aggarwal and H. Wang, editors, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 547–580. Springer US, 2010.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RS86] N. Robertson and P. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986.

[SA95] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proceedings of the 21st VLDB*, pages 407–419, 1995.

[SA96] R. Srikant and R. Agrawal. *Mining sequential patterns: Generalizations and performance improvements*. Springer, 1996.

[Sco12] J. Scott. *Social network analysis*. Sage, 2012.

[SLKB15] M. Sugiyama, F. Llinares-López, N. Kasenburg, and K. M. Borgwardt. Significant subgraph mining with multiple testing correction. In *Proceedings of the 15th SDM*, pages 100–207, 2015.

[SM97] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[SSS+93] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, and S. Arikawa. Finding alphabet indexing for decision trees over regular patterns: an approach to bioinformatical knowledge acquisition. In *Proceedings of the 26t HICSS*, pages 763–772, 1993.

[SVA97] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proceedings of the 3rd KDD*, pages 67–73, 1997.

[SVVL06] A. Siebes, J. Vreeken, and M. Van Leeuwen. Item sets that compress. In *Proceedings of the 6th SDM*, pages 393–404, 2006.

[SWG02] D. Shasha, J. T. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the 21st PODS*, pages 39–52, 2002.

[Sys82] M. M. SysŁ;o. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17(1):91 – 97, 1982.

[Tat08] N. Tatti. Maximum entropy based significance of itemsets. *Knowledge and Information Systems*, 17(1):57–77, 2008.

[TC12] N. Tatti and B. Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 25(1):34–66, 2012.

[TKS15] A. Terada, H. Kim, and J. Sese. High-speed westfall-young permutation procedure for genome-wide association studies. In *Proceedings of the 6th ACM-BCB*, pages 17–26, 2015.

[TOHTS13] A. Terada, M. Okada-Hatakeyama, K. Tsuda, and J. Sese. Statistical significance of combinatorial regulations. *Proceedings of the National Academy of Sciences*, 110(32):12996–13001, 2013.

[TTS13] A. Terada, K. Tsuda, and J. Sese. Fast westfall-young permutation procedure for combinatorial regulation discovery. In *Proceedings of the IEEE BIBM 2013*, pages 153–158, 2013.

[TV12] N. Tatti and J. Vreeken. Discovering descriptive tile trees. In *Proceedings of the ECML PKDD 2012*, pages 24–28, 2012.

[TWSY10] V. S. Tseng, C.-W. Wu, B.-E. Shie, and P. S. Yu. Up-growth: an efficient algorithm for high utility itemset mining. In *Proceedings of the 16th KDD*, pages 253–262, 2010.

[UAUA03] T. Uno, T. Asai, Y. Uchida, and H. Arimura. LCM: an efficient algorithm for enumerating frequent closed item sets. In *Proceedings of the FIMI '03*, 2003.

[UKA04] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the FIMI '04*, 2004.

[VL07] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[vLU14] M. van Leeuwen and A. Ukkonen. Fast estimation of the pattern frequency spectrum. In *Proceedings of the ECML PKDD 2014*, pages 114–129. Springer, 2014.

[VSKB10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010.

[VvLS11] J. Vreeken, M. van Leeuwen, and A. Siebes. Krimp: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[WF94] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[WHLT05] J. Wang, J. Han, Y. Lu, and P. Tzvetkov. TFP: An efficient algorithm for mining top-k frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):652–663, 2005.

[WHP03] J. Wang, J. Han, and J. Pei. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the 9th KDD*, pages 236–245, 2003.

[WM03] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explorations Newsletter*, 5(1):59–68, 2003.

[Wro97] S. Wrobel. An algorithm for multi-relational discovery of subgroups. In *Proceedings of the 1st PKDD*, pages 78–87. Springer, 1997.

[XCYH06] D. Xin, H. Cheng, X. Yan, and J. Han. Extracting redundancy-aware top-k patterns. In *Proceedings of the 12th KDD*, pages 444–453, 2006.

[YBYZ02] X. Yuan, B. P. Buckles, Z. Yuan, and J. Zhang. Mining negative association rules. In *Proceedings of the 7th ISCC*, pages 623–628, 2002.

[YH02] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 ICDM*, pages 721–724, 2002.

[YHA03] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large databases. In *Proceedings of the 3rd SDM*, pages 166–177, 2003.

[YHCL13] K.-J. Yang, T.-P. Hong, Y.-M. Chen, and G.-C. Lan. Projection-based partial periodic pattern mining for event sequences. *Expert Systems with Applications*, 40(10):4232–4240, 2013.

[YWY03] J. Yang, W. Wang, and P. Yu. Mining asynchronous periodic patterns in time series data. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):613–628, 2003.

[YYH04] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *Proceedings of the SIGMOD 2004*, pages 335–346, 2004.

[YZXS11] J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with knowledge from the physical world. In *Proceedings of the 17th KDD*, pages 316–324, 2011.

[YZZ+10] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL GIS*, pages 99–108, 2010.

[Zak01] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.

[ZH02] M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SDM*, volume 2, pages 457–473, 2002.

[ZKCY07] M. Zhang, B. Kaon, D. W. Cheung, and K. Y. Yip. Mining periodic patterns with gap requirement from sequences. *ACM Transaction on Knowledge Discovery from Data*, 1(2), 2007.

[ZPOL97] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd KDD*, pages 283–286, 1997.

Index

The contents of Chapter 6 are based on work published in IPSJ Transactions on Mathematical Modeling and Its Applications, vol.9(1), pp.32-42, 2016.